

A Real-Time, GPU-Based, Non-Imaging Back-End for Radio Telescopes

Alessio Magro

Supervisor: Dr. Kristian Zarb Adami

Co-supervisor: Dr. John Abela



University of Malta

Faculty of Science

27th September 2013



The research work disclosed in this publication is partially funded by the Strategic Educational Pathways Scholarship (Malta). This Scholarship is part-financed by the European Union – European Social Fund (ESF) under Operational Programme II – Cohesion Policy 2007-2013, “Empowering People for More Jobs and a Better Quality Of Life”.



Operational Programme II – Cohesion Policy 2007-2013
*Empowering People for More Jobs and a Better Quality of
Life*

Scholarship part-financed by the European Union
European Social Fund (ESF)
Co-financing rate: 85% EU Funds; 15% National Funds



Investing in your future

Faculty of Science

Statement of Authenticity

The undersigned declare that this dissertation is based on work carried out under the auspices of the Department of Physics by the candidate as part fulfilment of the requirements of the degree of Ph.D.

Alessio Magro
Candidate

Dr. John Abela
Co-supervisor

Dr. Kristian Zarb Adami
Supervisor

ABSTRACT

Since the discovery of Rapidly Rotating Transients (RRATs), interest in single pulse radio searches has increased dramatically. Due to the large data volumes generated by these searches, especially in planned surveys for future radio telescopes, such searches have to be conducted in real-time. This has led to the development of a multitude of search techniques and real-time pipeline prototypes. In this work we investigated the applicability of GPUs for such systems.

We have designed and implemented a scalable, flexible, GPU-based, transient search pipeline composed of several processing stages, including RFI mitigation, dedispersion, event detection and classification, as well as data quantisation and persistence. These stages are encapsulated as a standalone framework which can be used in offline mode, for processing archival data, as well as within an online application with additional real-time capabilities. The optimised GPU implementation of direct dedispersion achieves a speedup of more than an order of magnitude when compared to an optimised CPU implementation. We use a density-based clustering algorithm, coupled with a candidate selection mechanism to group detections caused by the same event together and automatically classify them as either RFI or of celestial origin. This setup was deployed at the Medicina BEST-II array in Italy, attached to an FPGA-based digital backend where several test observations were conducted. Finally, we calculate the number of GPUs required to process all the beams for the SKA₁-mid non-imaging pipeline.

We have also investigated the applicability of GPUs for beamforming, where our implementation achieves more than 50% of the peak theoretical performance. We also demonstrate that for large arrays, and in observations where the generated beams need to be processed outside of the GPU, the system will become PCIe bandwidth limited, with the attached GPUs spending most of the execution time waiting for I/O transfers. This can be alleviated by processing the synthesised beams on the GPU itself, and we demonstrate this by integrating the beamformer to the transient detection pipeline. We also analysed the beamforming computational requirements for SKA₁-low and SKA₁-mid, and demonstrate that GPUs are an inefficient architecture for this, with a very high running cost.

CONTENTS

Acknowledgements	xii
Publications	xiii
1. Introduction	1
1.1 Telescope Sensitivity	2
1.2 Effects of the Interstellar Medium	4
1.2.1 Dispersion	4
1.2.2 Scattering	6
1.2.3 Scintillation	8
1.3 Radio Transients	8
1.3.1 Classes of Radio Transients	10
1.4 Transient Detection Metrics	13
1.4.1 Transient Event Rates	15
1.4.2 Signal Combination Modes	16
1.5 Single Pulse Searches for Fast Transients	18
1.5.1 Single Pulse Searches S/N	20
1.5.2 Transient Surveys	21
1.6 Thesis Outline	23
2. Dispersion Removal Techniques and Implementation	25
2.1 Incoherent Dispersion Removal Techniques	26
2.1.1 Direct Dedispersion	26
2.1.2 Tree Dedispersion	27
2.1.3 Subband Dedispersion	31
2.1.4 Alternative Dedispersion Methods	31

2.2	Many-core Architectures	32
2.2.1	GPU Architecture	33
2.2.2	State of the Art Many-core Devices	35
2.2.3	Mapping algorithms to Many-core Architectures	36
2.3	GPU Framework	37
2.4	Direct Dedispersion GPU Implementation	39
2.4.1	Performance and Benchmarks	44
2.5	Coherent Dedispersion	48
2.5.1	GPU Implementation	49
2.6	Conclusion	52
3.	GPU-Based Transient Detection Pipeline	53
3.1	Transient Detection Pipelines	53
3.2	Pipeline Overview	54
3.3	RFI Mitigation	57
3.3.1	Bandpass Correction	58
3.3.2	Channel Thresholding	59
3.3.3	Spectrum Thresholding	61
3.3.4	RFI Mitigation Test	61
3.4	Dedispersion	62
3.5	Signal Post-processing	64
3.6	Detection and Candidate Selection	65
3.7	Data Persister	69
3.8	Pipeline Analysis	72
3.8.1	Clustering and Classification Evaluation	73
3.8.2	RFI Mitigation	76
3.8.3	Performance Benchmarks	78
3.9	Comparison with Other Work	79
3.10	Conclusion	82
4.	Deployment at the Medicina BEST-II Array	83
4.1	The BEST-II SKA pathfinder	83
4.2	BEST-II transient detection pipeline	86

4.2.1	Packet Receiver	88
4.2.2	Data Interaction Tool	91
4.3	BEST-II Observation Results	93
4.4	Conclusion	97
5.	GPU-Based Beamforming for Transient Discovery	100
5.1	Beamforming	101
5.2	Beamforming implementation on GPUs	103
5.2.1	Performance and Benchmarks	105
5.3	Pipeline Integration	109
5.4	Processing analysis of beamforming pipeline	111
5.5	Conclusion	113
6.	Applicability of GPUs to SKA₁	115
6.1	The Square Kilometre Array - Phase 1	115
6.1.1	SKA ₁ -low	116
6.1.2	SKA ₁ -mid	117
6.2	Technology Configuration	118
6.3	Transient Search Parameters	120
6.4	SKA ₁ -low station beamforming	122
6.5	SKA ₁ -low Station-level transient searching	125
6.6	SKA ₁ -mid CSP Beamforming	129
6.7	SKA ₁ -mid CSP Dedispersion	132
6.8	Conclusion	137
7.	Conclusion	138
	References	143
8.	SKA₁-low Station Transient Parameters	152

LIST OF FIGURES

1.1	Pulse dispersion	5
1.2	Pulse scattering	7
1.3	Transient phase space	9
1.4	Classes of rotating neutron stars	11
1.5	Candidate extragalactic bursts	12
1.6	AARTFAAC pipeline	14
1.7	Signal combination modes	17
1.8	Generic fast transient detection pipeline	19
2.1	Tree dedispersion algorithm	28
2.2	NVIDIA GK110 SMX block diagram	34
2.3	GPU framework workflow	38
2.4	Dedispersion memory access pattern	40
2.5	GPU memory access optimisation	41
2.6	Direct dedispersion test case	43
2.7	Configuration optimisation for direct dedispersion kernel	45
2.8	Direct dedispersion performance	46
2.9	Performance degradation relative to time shifts	47
2.10	Overlap-save method	50
3.1	Real-time transient pipeline schematic overview	55
3.2	Bandpass fitting example	60
3.3	Test case for RFI mitigation	62
3.4	RFI mitigation test case output	63
3.5	DBSCAN implementation performance	67
3.6	Effect of compression factor	70

3.7	μ -law 16-bit to 4-bit quantisation	71
3.8	Quantisation scaling performance	72
3.9	Matlab transient pipeline simulator	73
3.10	Clustering and classification accuracy	75
3.11	Pipeline detection accuracy	77
3.12	CRAFT backend implementation	81
4.1	The BEST-II array	84
4.2	BEST-II digital backend	86
4.3	BEST-II system architectural overview	87
4.4	BEST-II S-engine output packet format	88
4.5	Packet receiver schematic	90
4.6	Data interaction tool	92
4.7	BEST-II bandpass	94
4.8	Examples of BEST-II RFI events and their removal	95
4.9	PSR B0329+54 transiting through 8 synthesised beams	96
4.10	Integrated pulse profile for PSR B0329+54	97
4.11	Pipeline output for a test observation of PSR B0329+54	98
4.12	Positive and negative examples of cluster classification	99
5.1	Beamformer schematic	102
5.2	Configuration optimisation for beamforming kernel	106
5.3	Beamformer performance for varying number of beams	107
5.4	Beamformer performance for varying number of antennas	108
5.5	Integration of coherent beamformer in transient detection pipeline	110
5.6	Beam pattern for test setup	112
6.1	Ring-based beamforming	122
6.2	Resource utilisation for SKA ₁ -low station GPU-based processing	124
6.3	SKA ₁ -low station transient searching	127
6.4	SKA ₁ -low station processing costs	130
6.5	SKA ₁ -mid CSP ring-based beamforming	132
6.6	SKA ₁ -mid beamformer configuration	133

LIST OF TABLES

1.1	Transient Surveys	22
2.1	Comparison of current many-core architectures	35
3.1	Simulated data parameters for event detection accuracy test	74
3.2	Simulated data parameters for RFI thresholding accuracy test . . .	77
3.3	Transient detection pipeline timings	80
4.1	BEST-2 specifications	84
5.1	Beamforming pipeline timings	113
6.1	Key Specifications of SKA ₁ -low	117
6.2	Key Specifications of SKA ₁ -mid	118
6.3	NVIDIA Volta scalability measure	119
6.4	SKA ₁ -low station beamforming parameters	123
6.5	SKA ₁ -low station processing costs	125
6.6	SKA ₁ -low station transient parameters subset	129
6.7	Key Specifications of SKA ₁ -mid	131
6.8	SKA ₁ -mid survey parameters	135
6.9	Dedispersion requirements for SKA ₁ -mid	137
8.1	SKA ₁ -low station fast transient survey parameters	153

ACRONYMS

ADC	Analog to Digital Converter
ARTEMIS	Advanced Radio Transients Event Monitor and Identification System
ASM	All Sky Monitor
AVX	Advanced Vector Extensions
BEST	Basic Element for SKA Training
CASPER	Collaboration for Astronomy Signal Processing and Electronics Research
CMAC	Complex Multiply-Accumulate
CPU	Central Processing Unit
CRAFT	Commensal Real-time ASKAP Fast Transients
CSP	Central Signal Processing
CUDA	Compute Unified Device Architecture
DDR	Double Data Rate
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DM	Dispersion Measure
FFT	Fast Fourier Transform
FLOP	Floating Point Operation
FoV	Field of View
FoMSS	Figure of Merit Survey Speed
FPGA	Field Programmable Gate Array
FRB	Fast Radio Bursts

FWHM	Full Width Half Maximum
GbE	Gigabit Ethernet
GPU	Graphics Processing Unit
ILP	Instruction Level Parallelism
LOFAR	Low Frequency Array
PCI	Peripheral Component Interconnect
PFB	Polyphase Filter Bank
RFI	Radio Frequency Interference
RAM	Random Access Memory
RMSE	Root Mean Square Error
ROACH	Reconfigurable Open Architecture Computing Hardware
RRAT	Rapidly Rotating Transient
SKA	Square Kilometer Array
S/N	Signal-to-Noise
SPEAD	Streaming Protocol for Exchanging Astronomical Data
TDP	Thermal Design Power
UDP	User Datagram Protocol
XAUI	Ten Gigabit Attachment Unit Interface

ACKNOWLEDGEMENTS

I’m not really sure how I ended up doing a PhD, but I’m pretty certain that my supervisor, Dr Kristian Zarb Adami, had something to do with it. I have no regrets, it’s been an interesting and rewarding journey. Thanks for everything, you deserves endless hugging. A big thanks also goes to my co-supervisor, Dr. John Abela, who thinks that someday he’ll manage to eat more than I can.

Thanks to Aris Karastergiou for answering endless streams of questions about everything transient, and for setting in motion all the processes which would culminate in me writing this thesis. A big thanks to the dynamic SKA group at the University of Oxford and Oxford e-Research Centre, with people deserving special mention including: Ian Heywood, Sascha Schediwy, Stef Salvini, Fred Dulwich, Benjamin Mort, Wes Armour and especially Steve Rawlings, who will always be remembered in the hearts and minds of everyone who knew him. A big thanks goes to the “digital team”, namely Jack Hickish, Griffin Foster, Richard Armstrong and Danny Price, for making every trip to Oxford one to remember, and their immeasurable patience in dealing with my nagging.

I also thank the staff and support from Radiotelescopi di Medicina whom made the deployment at the BEST-II array possible, particularly Andrea Mattana, Federico Perini, Germano Bianchi, Giovanni Naldi, Jader Monari, Marco Bartolini, and Stelio Montebugnoli. Thanks also go to Andrew Siemion, Mark Wagner, Dan Werthimer and other people at CASPER, Berkeley, USA.

It’s been a blast working at the department of physics in the University of Malta. Special thanks goes to Prof. Charles Sammut, our beloved dean, the lovely Alison Darmanin and the rest of the department and faculty staff. Thanks to the founding member of the AstroMalta team: Jackson Said, Ian Fenech Conti, Adam Gauci, as well as the crazy students in the APRR. It’s been awesome!

I’d also like to thank in advance my examiners, Dr. Scott Ransom and Dr. Rob Fender. Last, but surely not least, I’d like to thank my family and friends for their endless and unquestioning support. I’m sure I forgot to mention a great number of people, but worry not, I thank you!

PUBLICATIONS

Refereed Publications

- **Magro A.**, Karastergiou A., Salvini S., Mort B., Dulwich F. and Zarb Adami, K. (2011). Real-time, fast radio transient searches with GPU de-dispersion. *Monthly Notices of the Royal Astronomical Society*, 17, 2642-2650.
- **Magro A.**, Hickish J. and Zarb Adami K.(2013). Multibeam GPU Transient Pipeline for the Medicina BEST-2 Array. *Journal of Astronomical Instrumentation*. 2 (1).
- Foster G., Hickish J., **Magro A.**, Price D. and Zarb Adami K. (2013). Implementation of a Direct-Imaging and FX Correlator for the BEST-2 Array. Accepted for publication in *Monthly Notices of the Royal Astronomical Society*

Conference Contributions

- Armour W., Karastergiou A., Giles M., Williams C., **Magro A.**, Zagkouris K., Roberts S., Salvini S., Dulwich F. and Mort B. (2012). A GPU-based survey for millisecond radio transients using ARTEMIS. In *ASP Conference Series* p. 33.
- **Magro A.** and Zarb Adami K. (2013). Pipeline Prototype for Fast Radio Transient Detection Using GPUs. 5th Workshop in Information and Communications Technology. SmartCity Malta
- Zheng H., Tegmark M. *et al.* (2013). Mapping our Universe in 3D with MITEoR. To be published in proceedings of 2013 IEEE International Symposium on Phased Array Systems and Technology

In Preparation

- Zarb Adami K. and **Magro A.**. (2013). GPU-Based Beamforming for Transient Discovery. *In prep.*

CHAPTER 1

INTRODUCTION

Radio Astronomy is a relatively modern field, which started with Karl G. Jansky in 1931. He worked as a radio engineer with Bell Telephone Laboratories in New Jersey. While studying radio frequency interference (RFI) from thunderstorms, he noticed a steady hiss type of static whose origin could not be attributed to weather systems. Rotating the antenna revealed that the static changed gradually, with a period of about 24 hours. He concluded that this static originated from the Milky Way [Jansky, 1933]. This field has grown exponentially since Jansky's time and has led to a substantial increase in astronomical knowledge, coupled with the construction of large radio instruments. The advent of radio interferometry enabled individual dishes and receiving elements to be combined into huge observatories, even spanning multiple continents, culminating in the current design and eventual deployment of the Square Kilometre Array, which will pose several data processing and transport challenges, some of which will be tackled in this thesis.

One of the more exotic areas of study in this field is the discovery and observation of several classes of radio transients, which are celestial objects that do not emit a steady amount of electromagnetic radiation. Their discovery is an excellent example of serendipity in astronomy. Pulsars were discovered in 1967 by Jocelyn Bell and Antony Hewish [Hewish et al., 1968] during a low-frequency survey of extragalactic sources that scintillate in the interplanetary plasma, while Gamma Ray Bursts (GRBs) were first identified in the late 1960s and early 1970s by the Vela satellite system which was launched to verify Soviet adherence to the nuclear test ban treaty [Klebesadel et al., 1973]. Both these examples demonstrate the necessity for current and future telescopes to explore uncharted regions of parameter space, in particular the need for higher sensitivity, finer time and spectral resolution and faster survey speed.

1.1 Telescope Sensitivity

In radio astronomy, radiation is generally measured as units of flux density, \mathcal{F} , which is the power per unit frequency interval that passes through a surface of unit area. Therefore the power density received by a radio receiver is

$$P = \frac{\mathcal{F}A_{\text{eff}}}{2} \quad (1.1)$$

where the factor of 2 reduction is due to the fact that we can only observe in the horizontal and vertical polarisation, and A_{eff} is the effective cross section of the antenna, which for wavelength λ and antenna gain G is defined as:

$$A_{\text{eff}} = \frac{\lambda^2 G}{4\pi} \quad (1.2)$$

For the small fluxes encountered in radio astronomy it is convenient to define the unit Jansky (Jy), where $1 \text{ Jy} = 10^{-26} \text{ W Hz}^{-1} \text{ m}^{-2}$. For comparison, the Sun, which is the brightest natural radio emitter in the sky, has a flux density of about 4 MJy when observed at 10 GHz.

A radio receiver measures the power density P picked up by an antenna, which can be compared with the thermal noise produced by a bandwidth limited resistor of a given temperature T :

$$P = k_B \Delta \nu T \quad (1.3)$$

where k_B is the Boltzmann constant, equal to $1.38 \times 10^{-23} \text{ W s/K}$ [Johnson, 1928]. The system temperature of the telescope, T_{sys} , is a measure of the instantaneous noise contribution to a measurement from the telescope receiver chain, the sky, which is at a constant temperature itself, and any other sources of noise in the telescope. Using equation 1.3, the system temperature can be defined as

$$T_{\text{sys}} = \frac{P}{k_B \Delta \nu} \quad (1.4)$$

Using this definition, the noise of a measurement from a telescope is then determined by the ideal radiometer equation (see [Burke and Graham-Smith, 2010, chapter 3]:

$$\Delta T = \frac{T_{\text{sys}}}{\sqrt{\Delta \nu \Delta t}} \quad (1.5)$$

The sensitivity to a point source can be expressed by considering the flux

collecting area of a telescope A_{eff} , which yields an expression for the noise on a measurement of flux density ΔS from a point source

$$\Delta S = \frac{2k_B T_{\text{sys}}}{A_{\text{eff}} \sqrt{\Delta \nu \Delta t}} \quad (1.6)$$

This shows the critical dependence of point source sensitivity to $A_{\text{eff}}/T_{\text{sys}}$, and thus this ratio is often quoted when specifying the top level requirements of a telescope. The sensitivity of a telescope to a point source can always be improved by building a more directive antenna (or antenna array) with larger effective area. If β is the minimum S/N for a source to be considered detected, then the minimum detectable flux density, S_{min} , is

$$S_{\text{min}} = \frac{2k_B \beta}{A_{\text{eff}}} \frac{T_{\text{sys}}}{\sqrt{N_p \Delta \nu \Delta t}} \quad (1.7)$$

where N_p is the number of orthogonal polarisations averaged, and can have a value of either 1 or 2. In the case of pulsars, the data obtained from the receiver can be folded at the pulse period to enhance the S/N of the pulse, and S_{min} will then be dependent on the pulse period, duty cycle and integration time (interested readers may refer to [Bhattacharya, 1998] for an analytic analysis of how S_{min} is affected).

The intensity, or brightness temperature, I , is also used as a measure of radiation. This is the power per unit frequency interval passing through a surface of unit area and from a cone of unit solid angle. The solid angle Ω measures the fraction of the entire sky which is covered by the source, the whole area having a solid angle of 4π . The unit of intensity is $\text{W Hz}^{-1} \text{ m}^2 \text{ sr}^{-1}$, where sr stands for steradian, meaning “per unit angle”. Flux and intensity are related by

$$\mathcal{F} = I \Omega \quad (1.8)$$

The intensity along a line of sight in a vacuum does not change, irrespective of the distance from the source. It only changes if there are additional light sources, or if radiation is absorbed by intervening material. However, the flux received from a source decreases with the square of the distance.

1.2 Effects of the Interstellar Medium

Before signals from astrophysical sources reach the Earth they pass through the interstellar medium (ISM) and, for the case of signals originating from extragalactic sources, the intergalactic medium (IGM). This is a cold, ionised plasma which causes these signals to experience a frequency-dependent index of refraction as they propagate through it. Following [Lorimer and Kramer, 2005], the refractive index is

$$\mu = \sqrt{1 - \frac{f_p^2}{f^2}} \quad (1.9)$$

where f is the observing frequency and the plasma frequency

$$f_p = \sqrt{\frac{e^2 n_e}{\pi m_e}} \simeq 8.5 \text{ kHz} \left(\frac{n_e}{\text{cm}^{-3}} \right)^{\frac{1}{2}} \quad (1.10)$$

where n_e is the electron number density, and e and m are the charge and mass of an electron respectively. For the ISM, $n_e \simeq 0.003 \text{ cm}^{-3}$ and $f_p \simeq 1.5 \text{ kHz}$.

1.2.1 Dispersion

From equation 1.9, $\mu < 1$, therefore the group velocity of a propagating wave $v_g = c\mu$ is less than the speed of light c . Therefore, the propagation of a radio signal along a path of length d from a source to Earth will be delayed in time with respect to a signal of infinite frequency by an amount

$$t = \left(\int_0^d \frac{dl}{v_g} \right) - \frac{d}{c} \quad (1.11)$$

Substituting $v_g = c\mu$ and noting $f_p \ll f$ to approximate μ , we get

$$t = \frac{1}{c} \int_0^d \left[1 + \frac{f_p^2}{2f^2} \right] dl - \frac{d}{c} = \frac{e^2}{2\pi m_e c} \frac{\int_0^d n_e dl}{f^2} \equiv \mathcal{D} \times \frac{\text{DM}}{f^2} \quad (1.12)$$

where the dispersion measure

$$\text{DM} = \int_0^d n_e dl \quad (1.13)$$

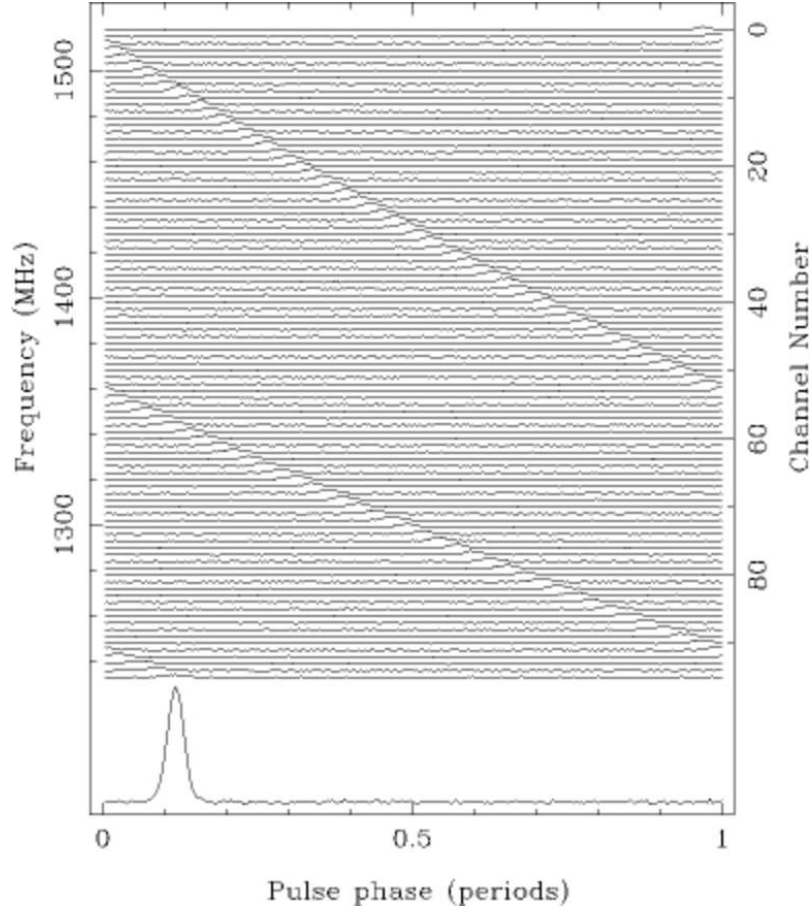


Figure 1.1: Pulse dispersion shown in the Parkes observation of the 128 ms pulsar B1356-60, with a dispersion measure of $295 \text{ cm}^{-3} \text{ pc}$. Source: [Lorimer and Kramer, 2005, figure 1.8]

is generally expressed in pc cm^{-3} , and the dispersion constant ([Lorimer and Kramer, 2005])

$$\mathcal{D} = \frac{e^2}{2\pi m_e c} = (4.148808 \pm 0.000003) \times 10^3 \text{ MHz}^2 \text{ pc}^{-1} \text{ cm}^3 \text{ s} \quad (1.14)$$

where the uncertainty in \mathcal{D} is determined by the uncertainties in e and m_e .

The above process has the effect of delaying the signals in a frequency dependent manner, whereby signals at lower frequencies are delayed more than those at higher frequencies. We can calculate the time delay Δt between two signals f_1 and f_2 with the dispersion relationship equation:

$$\Delta t = k_{\text{DM}} \cdot \text{DM} \cdot (f_1^{-2} - f_2^{-2}) \quad (1.15)$$

where $k_{\text{DM}} = \mathcal{D} \times 10^3$. In practice the approximation

$$k_{\text{DM}} = \frac{1}{2.41 \times 10^{-7}} \times 10^3 \quad (1.16)$$

is used. Figure 1.1 demonstrates the dispersion effect on a pulse from the 128 ms pulsar B1356-60, observed with the Parkes Radio Telescope, having a DM of 295 pc cm⁻³. The DM value for any source can be determined by measuring the phase as a function of frequency, which allows the distance to the object to be calculated by numerically integrating equation 1.13, assuming one has a model for the Galactic electron density distribution.

When observing radio sources this dispersive effect needs to be corrected for, in order to avoid excessive smearing in the time series and the consequent loss in signal-to-noise (S/N). This can be performed by using a dedispersion algorithm, some of which are discussed in detail in sections 2.1 and 2.5. These techniques require the source’s DM value to correct for dispersion, however, when searching for new sources, such as in blind transient surveys, the distance to any potential discovered source is unknown, and so this procedure needs to be repeated for many trial dispersion measures. This is a computationally expensive task, where a simple approach involves summing all the frequency channels across the observing band ($\sim 10^3$ frequency channels) for a number of trial DMs ($\sim 10^4$, typically) for every time sample, where modern sampling intervals are of the order of μs . This task is even more challenging when multiple beams need to be processed (for interferometric arrays or multiple-feed dishes).

1.2.2 Scattering

The electron density in the ISM is not homogeneous but shows variations on a wide range of scales. This results in temporal variations in dispersion measure, which also distort and scatter the pulse shape. Multi-path propagation temporally broaden narrow pulses emitted from a radio source and are observed as a one-sided exponential function with a scattering timescale τ_s . More complicated pulse shapes appear as a convolution of the pulse with the exponential. This effect is depicted in figure 1.2, where pulse profiles of PSR B1831-03, observed at five different frequencies with the Lovell telescope and the GMRT, show the increasing effects of scattering at lower frequencies. Several authors have investigated a possible empirical relationship between the scattering timescale and the dispersion measure.

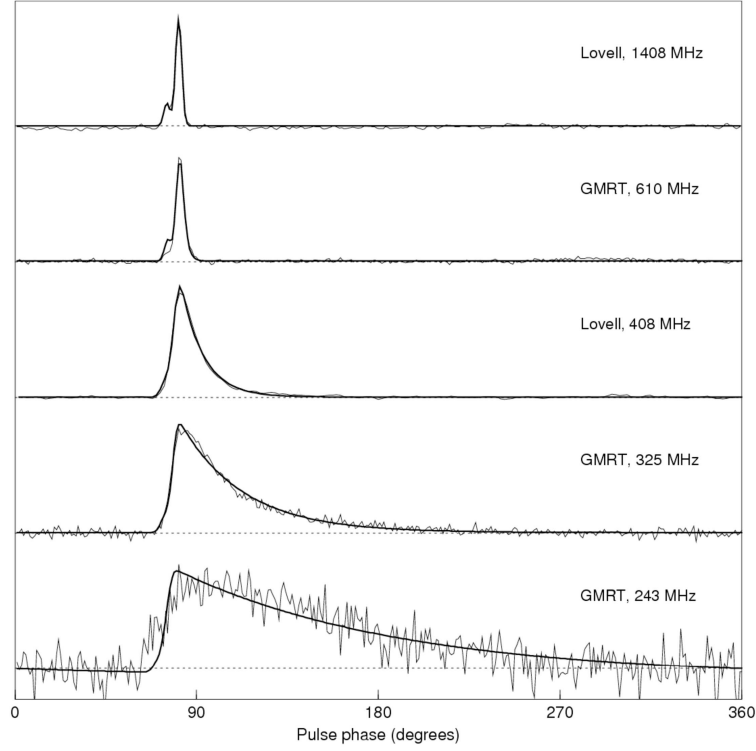


Figure 1.2: Pulse profiles for PSR B1831-03 observed at five different frequencies with the Lovell telescope and the GMRT, clearly showing the increasing effect of scattering at lower frequencies. The solid lines show exponential fits to the data. Source: [Lorimer and Kramer, 2005, figure 1.11]

We reproduce the empirical fit by [Bhat et al., 2004] here:

$$\log \tau_s = -6.46 + 0.154 \log(\text{DM}) + 1.07(\log(\text{DM}))^2 - 3.86 \log f \quad (1.17)$$

where τ_s is in ms and f in GHz. The scattering effect limits the sensitivity of transient surveys, which causes higher DM pulses to be stretched by a substantial amount along the time series, significantly reducing the S/N of detected pulses, especially for low frequency telescopes.

This relationship applies to sources in the Milky Way. When observing extragalactic source τ_s will be smaller, even for large DM values, although the distribution of the intergalactic medium (IGM) is not yet well understood. Recently [Lorimer et al., 2013] rescaled this relationship for extragalactic fast transients, based on measured properties of the bursts discovered by [Lorimer et al., 2007], [Keane et al., 2012] and [Thornton et al., 2013], where the leading term (-6.5) is replaced by -9.5. They however state that since only one measurement of the

scattering timescale has been made so far, it is likely that this is an upper limit to the average amount of scattering as a function of DM.

It should be noted that the observed τ_s for a given DM can deviate from the predicted value by up to two orders of magnitude. Following [Hassall et al., 2013], assuming the unscattered pulse is a step function with intrinsic width τ_0 and a peak flux density S_{v0} , then its intrinsic fluence is $\mathcal{F} = S_{v0}\tau_0$. As the pulse is broadened by scattering, the fluence at a given frequency F_v is given by:

$$\mathcal{F}_v = S_v \sqrt{\tau_0^2 + \left(\int_0^\infty e^{-t/\tau_s} dt \right)^2} = S_v \sqrt{\tau_0^2 + \tau_s^2} \quad (1.18)$$

where S_v is the observed peak flux density at frequency v . Assuming that the fluence is conserved by scattering, the peak flux density is given by

$$S_v = \frac{\mathcal{F}}{\sqrt{\tau_0^2 + \tau_s^2}} \quad (1.19)$$

1.2.3 Scintillation

Relative motion between the source, the scattering medium and the observer leads to the phenomenon of interstellar scintillation (ISS), which manifests itself as intensity variations on various timescales. This effect is very similar to what causes stars to twinkle in the night sky. ISS is not taken into consideration in the work presented in this thesis, however interested readers can refer to [Lorimer and Kramer, 2005, section 4.2] for a more thorough description of this phenomenon.

1.3 Radio Transients

The exploration of the transient universe is an exciting and rapidly growing area in radio astronomy. Known transient phenomena range in time scales from sub-nanosecond to years, yielding a rich diversity in their underlying physical processes and acting as probes to physical and astrophysical phenomena in extreme environments, where their density, temperature, pressure, velocity, gravitational and magnetic fields far surpass the capabilities of any Earth-based laboratory. These transient phenomena are thought to be likely locations of cataclysmic or dynamic events, thus providing enormous potential to uncover a wide range of new physics.

Most of the objects have been discovered through a limited number of surveys

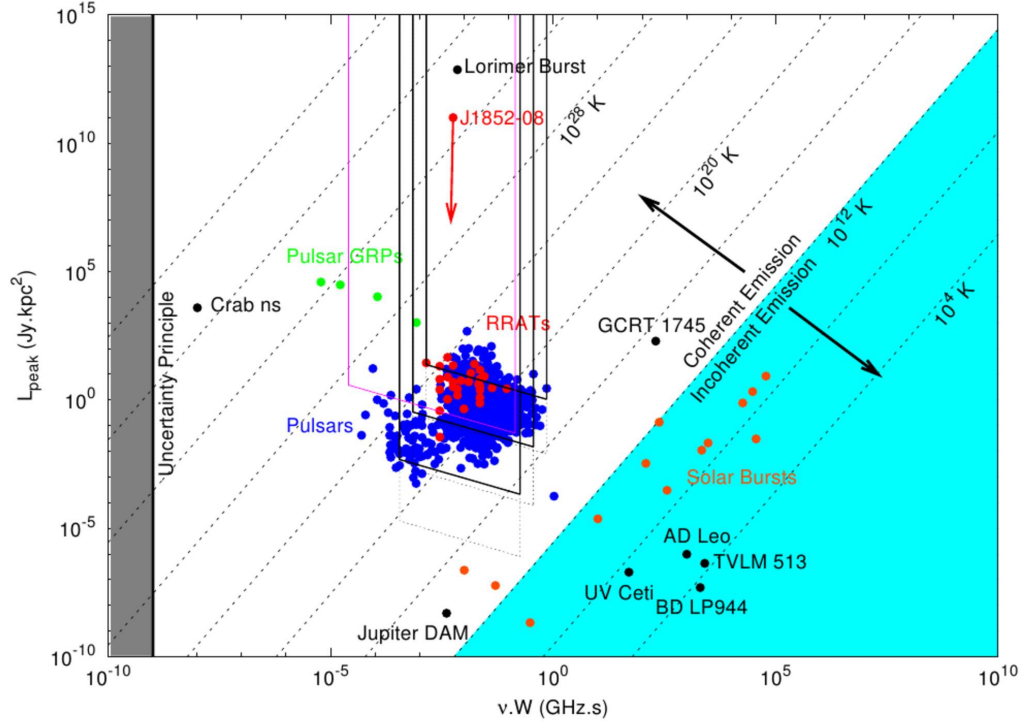


Figure 1.3: The transient ‘phase space’ with known sources identified, plotting the pseudo-luminosity $L = SD^2$ versus vW , where S is flux density, D is distance, v is observing frequency and W is pulse width, with a representative number of sources plotted, including pulsars [Hobbs et al., 2003], RRATs [McLaughlin et al., 2006], pulsar giant radio pulses [Cognard et al., 1996, Romani and Johnston, 2001], flare stars [Bastian, 1994, Richards et al., 2003, Osten and Bastian, 2008], auroral radio emission from the Sun and planets [Dulk, 1985, Zarka, 1998], GCRT 1745-3009 [Hyman et al., 2006] and the Lorimer burst [Lorimer et al., 2007]. Source: [Keane et al., 2011]

and serendipitous discoveries, meaning that the time domain sky has, up till recent times, been sparsely explored, suggesting that there is much to be found and studied. Over the next decade, the “new generation of radio telescopes”, with improved sensitivity, wider fields of view, higher survey speed and flexible digital signal processing backends will be able to explore radio transient parameter space more comprehensively and systematically. This parameter space, also referred to as the transient phase space, is extensive. Transients have been detected, and are predicted for, all radio wavelengths, across a wide range of timescales, and may originate from nearly all astrophysical environments including the solar system, star-forming regions, the Galactic centre, and beyond our host galaxy.

The transient population is generally divided into two types, although the exact cutoff currently relies on the snapshot time of current radio instruments,

which is related to the shortest time at which an image can be generated. Transients can be classified as either “fast” or “slow”. Fast transients have a duration of less than ~ 1 s and arise from coherent emission. These include pulsar and neutron star phenomena, solar bursts, flare stars and annihilating black holes. Due to the nature of these signals, real-time data processing is a challenge, and severe effects caused by the Inter Stellar Medium (ISM), especially at low frequencies, where the field of view is highest, and large bandwidths, need to be corrected for. Slow transients arise from incoherent (synchrotron) emission, and can be defined as those transients with time scales longer than the time it takes to image the relevant region of the sky. Known source classes include novae, Active Galactic Nuclei (AGN) and Gamma-ray bursts (GRB). Figure 1.3 provides an illustration of the transient phase space, as depicted by [Keane et al., 2011], with known sources identified.

1.3.1 Classes of Radio Transients

Classes of transients are diverse, ranging from nearby stars to objects at cosmological distances, and touching upon nearly every aspect of astronomy, astrophysics and astrobiology. These classes include, but are not limited to:

(i) *Neutron Stars*. Neutron stars are the most populous member of the transient radio phase space, and are thus the most well studied. They are highly compact stars with radii of ~ 10 km and masses of $\sim 1.4 M_{\odot}$, with extremely strong gravity and magnetic fields, 10^{12} G being typical (see [Shapiro and Teukolsky, 1983]). Pulsars are those rapidly rotating neutron stars which emit an apparently steady, narrow beam of emission. This emission seems to originate from fixed regions on/above the neutron star surface so that, modulated by the star’s rotation, the beams can sweep across our line of sight and be detected at Earth. Thus observers see a pulse of emission per rotation. Rotating Radio Transients (RRATs) [McLaughlin et al., 2006] are those rapidly rotating neutron stars whose emission does not seem to be steady and they are usually seen to be “off”. These are generally single, dispersed bursts of duration 2 - 30 ms, repeating with the same DM and having a recurrence time of 4 min - 3 hours. In addition to their pulsed emission, neutron stars can be transient in other respects. There are pulsars known in eclipsing binaries, where the orbit plane of a binary pulsar lies so close to the line of sight that the components undergo mutual eclipses, as well as “nulling” pulsars, which turn on and off for weeks at a time. Figure 1.4 shows a series of pulsars

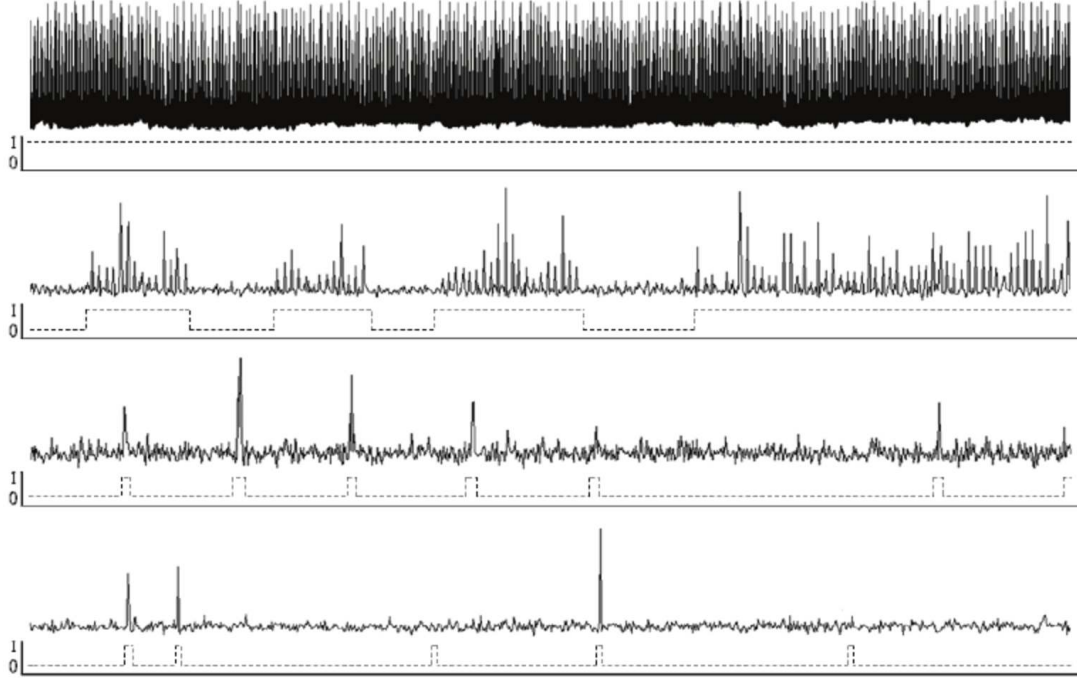


Figure 1.4: A series of pulsars exhibiting a range of emission activity timescales. Top to bottom: Vela, PSRs J1646-6831, J1647-36, J1226-32. All panels are of equal duration. The binary scale below each timeline shows an estimated representation of the null/emission state. Source: [Burke-Spolaor, 2012, figure 2]

exhibiting a range of emission activity timescales.

(ii) *Pulsar Giant Pulses.* While all pulsars show pulse-to-pulse intensity variations, some pulsars emit so-called giant pulses, with strengths 100 or even 1000 times the mean pulse intensity. The Crab was the first pulsar found to exhibit this phenomenon, and giant pulses have since been detected from numerous other pulsars, for example [Cognard et al., 1996, Romani and Johnston, 2001, Johnston and Romani, 2003]. Pulses with flux densities of order 10^3 Jy at 5 GHz and with durations of only 2 ns have been detected from the Crab [Hankins et al., 2003]. These “nano-giant” pulses imply brightness temperatures of 10^{38} K, which are considered to be amongst the most luminous emissions from any astronomical object. In addition to being probes of particle acceleration in pulsar magnetosphere, giant pulses may serve as probes of the local intergalactic medium [McLaughlin and Cordes, 2003].

(iii) *Fast Radio Bursts.* In 2007, the detection of a 30 Jy, 5 ms duration, highly dispersed (DM of $375 \text{ cm}^{-3} \text{ pc}$) burst, detected in 3 independent beams at Parkes, was reported by [Lorimer et al., 2007] (figure 1.5a), thought to be of

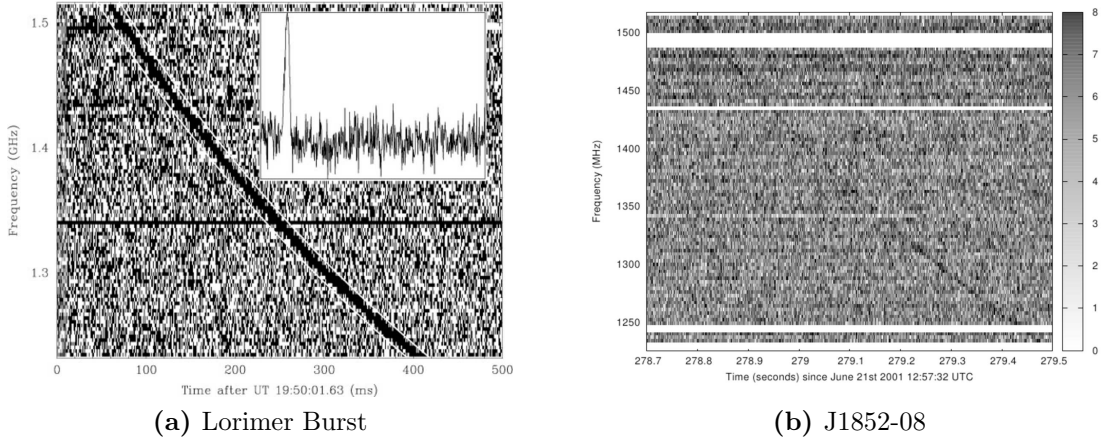


Figure 1.5: Waterfall plots for two candidate extragalactic bursts. (Left) The Lorimer Burst, a 30Jy dedispersed burst of duration less than 5ms located 3° south of the centre of the Small Magellanic Cloud (SMC). It was found by [Lorimer et al., 2007] in archival survey data. (Right) A 400 mJy pulse from J1852-08, discovered in the Parkes Multi-beam Pulsar Survey (PMPS) by [Keane et al., 2012] with a DM of $746 \text{ cm}^3 \text{ pc}$ and pulse width of 7.8ms.

extragalactic origin. A second burst was detected by [Keane et al., 2011] (figure 1.5b), with a DM of $745 \text{ cm}^{-3} \text{ pc}$. Recently, [Thornton et al., 2013] reported an additional 4 millisecond-duration radio transients, all more than 40° from the Galactic plane, also of extragalactic origin. Currently no temporally coincident X- or γ ray signatures were identified in association with these bursts. These recent discoveries have spurred an increased interest in conducting surveys for highly dispersed, single pulses signals, with recent studies suggesting that the low frequency component of the SKA₁ could find an extragalactic burst every hour [Hassall et al., 2013].

(iv) *Flare Stars, Brown Dwarfs and Extrasolar Planets.* Active stars and star systems have been known to produce radio flares attributed to particle acceleration from magnetic field activity [Güdel, 2002]. Flares from late-type stars and brown dwarfs have also been discovered [Berger et al., 2001, Hallinan et al., 2007], in some cases with periodicities indicative of rotation. Finally, Jupiter is bright below 40 MHz, and many stars with “hot Jupiters” show signatures of magnetic star-planet interactions [Shkolnik et al., 2005], suggesting that extrasolar planets may also be radio sources [Zarka, 2007], indicating that searches for exoplanets can be conducted with next generation radio telescopes.

(v) *Radio Supernovae and GRBs.* Frequent observation of a large area of sky, as made possible by recent and future radio telescopes, can be used to

find GRBs and supernovae that emit in the radio regime, as well as to follow up on such transients detected at other wavelengths. Multi-wavelength, multi-epoch observations (eg [Cenko et al., 2006]) can provide information on progenitors, the surrounding medium and models of GRB energetics and beaming.

(vii) *Annihilating Black Holes*. Annihilating black holes are predicted to produce radio bursts [Rees, 1977]. Advances in γ -ray detectors has renewed interest in possible high-energy signatures from primordial black holes [Dingus et al., 2002, Linton et al., 2006]. Observations at the extremes of the electromagnetic spectrum are complementary as radio observations attempt to detect the pulse from an individual black hole, while high-energy observations generally search for the integrated emission.

(viii) *Gravitational wave events*. The progenitors for gravitational waves may generate associated electromagnetic signals or pulses. For example, the in-spiral of a binary neutron star system may produce electromagnetic pulses, both at high energies and in the radio due to the interaction of the magnetosphere of the neutron stars [Hansen and Lyutikov, 2001]. More generally, the combined detections of both electromagnetic and gravitational wave signals may be required to produce localisations and understanding of the gravitational wave emitters [Bloom et al., 2009].

(ix) *Extraterrestrial transmitters*. While none are known, searches for extraterrestrial intelligence (SETI) have found non-repeating signals that are otherwise consistent with expected signals from an ET transmitter. [Cordes et al., 1997] show how ET signals could appear transient, even if intrinsically steady. More recently, several searches were conducted towards Kepler’s field of view, which to date has confirmed 151 planets, with more than 3,000 candidates still being analysed.

1.4 Transient Detection Metrics

Phased-arrays with very wide fields of view can essentially image the entire sky in a time T_c , which is the fastest possible correlator dump time. Figure 1.6 shows a schematic overview of the Amsterdam-ASTRON Radio Transient Facility And Analysis Center (AARTFAAC) [Prasad and Wijnholds, 2012], which aims to implement a near real-time, 24x7 All-Sky Monitor (ASM) for LOFAR which will be capable of monitoring low frequency radio transients over most of the sky locally

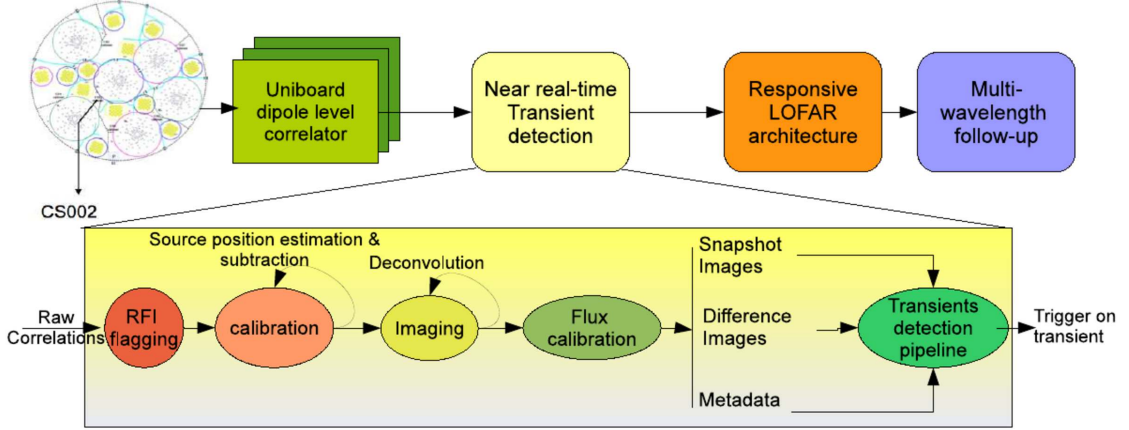


Figure 1.6: The main components of the AARTFAAC ASM. The correlator outputs are first passed through an RFI excision stage which generates appropriate RFI masks. The correlations are then calibrated and imaged with low latency, with the output images passing through a flux calibration stage and then through the Transients Pipeline. This carries out source extraction, source association, and the generation of light curves from existing observations for transient detection. Source: [Prasad and Wijnholds, 2012, figure 2].

visible to LOFAR, at timescales ranging from seconds to several days. Developments in millisecond imaging, such as [Law et al., 2011], are constantly reducing the snapshot time for these type of surveys.

Fast transients are those which cannot be well sampled through imaging surveys, unless they are very frequent and there is tolerance for a low completeness coefficient. Rare, fast transients are better sampled through staring observations of large solid angles. We follow [Cordes, 2009] in defining a survey metric based on the amount of volume surveyed by a radio instrument. Surveys that involve time-variable sources need to take into account the event rates and durations of transient sources, as well as sensitivity requirements. Slow transients are those for which the sky may be sampled by raster scanning, where the telescope beam with beam size Ω_i scans a patch of sky of total solid angle Ω_s in a time T_s . This scan is then repeated for the duration of the survey. The dwell time per sky position, that is the time spent pointing to the same patch of sky, is $\tau = (\Omega_i/\Omega_s)T_s$.

For blind searching, the rate of sky coverage $\dot{\Omega}$ ($\text{deg}^2 \text{s}^{-1}$) needs to be maximised, while also achieving the desired search depth, characterised by the maximum detection distance D_{max} . The yield of a survey, or the number of events detected per unit time, involves the product of source number density n_s and

search volume $V_{\max} = \frac{1}{3}\Omega_s D_{\max}^2$. If the survey of the solid angle Ω_s is conducted in a time T_s , the resulting search volume yields a combination of parameters similar to that obtained by calculating the survey speed (SS), which is Ω_s/τ . These two approaches lead to the figure of merit for steady sources (FoMSS), which can be defined as (see [Cordes, 2009, appendix a]):

$$\text{FoMSS} = B \left(\frac{N_{\text{FoV}} \Omega_{\text{FoV}}}{N_{\text{sa}}} \right) \left(\frac{f_c A_{\text{eff}}}{m T_{\text{sys}}} \right)^2 \quad (1.20)$$

where B is the bandwidth, N_{FoV} is the number of fields of view (or pixels) for each antenna, Ω_{FoV} is the solid angle for each FoV, N_{sa} is the number of subarrays into which the array is divided, which are assumed to be equal in size and pointed in non-overlapping directions, f_c is the fraction of the total effective area A_{eff} usable in the survey, m is the threshold S/N in the survey for which an event is considered to be detected and T_{sys} is the system temperature.

FoMSS is relevant to surveys of sources that are homogeneously distributed within the spatial domain, that are steady, standard candles. A more general metric which applies to transients is defined by [Cordes, 2009]:

$$\text{FoMTS} = \text{FoMSS} \times \mathcal{K}(\eta W, \tau/W) \quad (1.21)$$

where $\mathcal{K}(\eta W, \tau/W)$ accounts for the fact that sources with burst time W may only be on for a fraction of the dwell time τ , and the burst event time follows a Poisson distribution with rate η .

1.4.1 Transient Event Rates

The detection rate for fast transients has also been well studied by [Macquart et al., 2010] and [Colegate and Clarke, 2011]. Here we present a brief overview of their results, the interested reader can refer to the original documents.

The main goal of transient surveys is to maximise the number of events detected, which primarily depends on the search strategy employed on the telescope. Given each search strategy has a different processing cost, we can define a new FoM, the event rate per unit cost $\mathcal{R}_{\text{cost}^{-1}}$, which can be a more comprehensive FoM than survey speed. Since signal and search processing costs are architecture specific, the “event rate per beamformed and searched”, referred to as $\mathcal{R}_{\text{beam}^{-1}}$, can be used to generalise the problem, and is based on the rate of transient events detectable in a volume of sky. It is assumed that $\mathcal{R}_{\text{cost}^{-1}} \propto \mathcal{R}_{\text{beam}^{-1}}$, where cost

increases linearly with number of beams processed.

For fast transient searches, one volume of sky is considered as likely as another to contain transient events, and each time a volume is revisited there is new detection potential. This enables the deployment of digital backends for commensal surveys (sometimes called “piggy-back” surveys), where the survey does not point the telescope beams itself but rather branches off the data stream generated by other observations from the signal processing chain and performs a transient search on it. Based on [Macquart et al., 2010], [Colegate and Clarke, 2011] define the event rate as

$$\mathcal{R}_v = \rho \frac{\Omega_p}{4\pi} V_{\max} \text{ events s}^{-1} \quad (1.22)$$

where V_{\max} is the maximum volume out to which an object is detectable and Ω_p is the processed FoV, which is the product of the number of beams formed (N_{beam}) and the FoV of each beam. Thus the event rate per beam is

$$\mathcal{R}_{\text{beam}^{-1}} = \frac{\mathcal{R}_v}{N_{\text{beam}}} \quad (1.23)$$

An extragalactic survey is described as a search for a homogeneously distributed population of isotropically emitting fast transients of fixed intrinsic luminosity. For such a population, the event rate is

$$\mathcal{R}_v = \frac{1}{3} \rho \Omega_p \left(\frac{W_i}{W} \right)^{\frac{3}{4}} \left(\frac{\mathcal{L}_i}{4\pi S_{\min}} \right)^{\frac{3}{2}} \text{ events s}^{-1} \quad (1.24)$$

where ρ (events s⁻¹ pc⁻³) is the event rate density, \mathcal{L}_i (Jy pc²) is the intrinsic luminosity of the population, W_i is the intrinsic width, W is the observed pulse width and S_{\min} is the minimum detectable flux density of the telescope, defined in equation 1.7, with integration time $\tau = W_i$. The $(W_i/W)^{3/4}$ term approximates the loss in S/N due to pulse broadening.

1.4.2 Signal Combination Modes

Incoherent addition of an array of N_0 elements increases the sensitivity by a factor of $\sqrt{N_0}$ over a single element while retaining its full FoV. Forming N_{b-0} station beams linearly increases the FoV. To even further increase the FoV, N_{sa} subarrays can be incoherently combined, where subarrays are pointed in a different directions, increasing the FoV by a factor N_{sa} , but only increasing the sensitivity by a factor $\sqrt{N_{0/\text{sa}}}$ over a single element, where $N_{0/\text{sa}}$ is the number of elements per subarray.

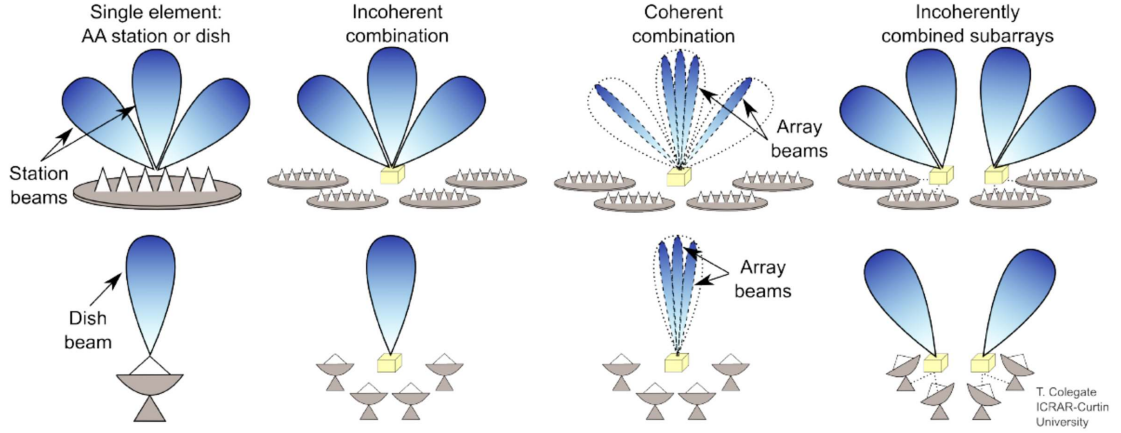


Figure 1.7: Signal combination modes, resultant beam patterns and beam terminology for dishes and aperture array. Beams sizes are not to scale. Source: [Colegate and Clarke, 2011, figure 2]

Fly’s eye pertains to the case where $N_{0/\text{sa}} = 1$.

Sensitivity of the coherent combination of an array of N_0 elements is higher than incoherent combination and subarraying as it increases proportional to N_0 . However the FoV of the array beam, Ω_{arr} , is much smaller and is proportional to D_{arr}^{-2} , where D_{arr} is the diameter of the array of elements being combined. The FoV can be linearly increased by forming $N_{b-\text{arr}}$ array beams. Applying these relationships to equation 1.24 gives the total event rate for each signal combination mode (see [Colegate and Clarke, 2011, section 4]):

$$\mathcal{R}_v = \frac{1}{3}\rho \left(\frac{W_i N_p \Delta v \tau}{W} \right)^{\frac{3}{4}} \left(\frac{\mathcal{L}_i A_{\text{eff}}}{4\pi\sigma 2k_B T_{\text{sys}}} \right)^{\frac{3}{2}} \mathcal{M} \quad (1.25)$$

$$\mathcal{M} = \begin{cases} N_{b-0} \Omega_0 N_0^{\frac{3}{4}} & \text{Incoherent combination} \\ N_{b-\text{arr}} \Omega_{\text{arr}} N_0^{\frac{3}{2}} & \text{Coherent combination} \\ N_{\text{sa}}^{\frac{1}{4}} N_{b-0} \Omega_0 N_0^{\frac{3}{4}} & \text{Subarraying} \end{cases} \quad (1.26)$$

where N_p is the number of polarisations summed, Δv is the processed bandwidth, τ is the post-detection integration time and σ is the S/N ratio required for event detection. The coherent combination mode is more effective if the elements are closely spaced, thus having a higher filling factor, resulting in a larger array beam FoV. Figure 1.7 shows a schematic illustration of the various combination modes mentioned in this section.

1.5 Single Pulse Searches for Fast Transients

Since the discovery of RRATs, interest in single radio pulse searches has increased dramatically. Single pulse detection extensions have been incorporated into current pulsar surveys, and several archival pulsar data have been reanalysed for detecting such events. Due to the large data volumes generated by these searches, especially in planned surveys for future radio telescopes, too large to store cost-effectively, such searches have to be conducted in real-time, on a data stream which is a continuous observation of the sky. Such systems are referred to as pipelines, whereby the input antenna voltages pass through a chain of processing elements, each performing a specific task. In order to be able to further process positive candidate events offline, after the observation has been conducted, a rolling buffer is required, which stores a small period of the data as it is being observed. This buffer is sometimes referred to as a transient buffer board (TBB). The specific implementation of such a pipeline depends on the target or expected source population, as well as the predicted cost and performance factors of components in the processing pipeline. Figure 1.8 shows a generic fast transients pipeline, the components of which are described below.

(i) *Signal reception*: Radio signals collected by radio dishes or aperture arrays are digitised and subsequently quantised, to reduce the required transmission data rate. This processing can be performed on a digital backend or conventional compute cluster. Data transfer between this backend and hardware running the transients pipeline generally occurs via high-throughput network links, at which point the packetised data are decoded, assembled into processable buffers and prepared for analysis.

(ii) *Signal Combination*: Signals from dish or station beams can be combined coherently, incoherently or not at all, as discussed in the previous section. Incoherent (phase insensitive) combination sums the detected signals from antennas pointing in the same direction. Coherent combination forms phased or tied array beams, where voltages measured at each antenna are phase aligned towards a specific direction in the sky. Smaller groups of antennas (subarrays) can be incoherently combined and each subarray pointed in a different direction.

(iii) *RFI Mitigation*: Radio Frequency Interference (RFI) mitigation is one of the major problems in transient surveys, especially for radio telescope which are close to urban centres. Discriminating between astronomical and terrestrial signals is a challenge, and several techniques have been investigated for this purpose. This

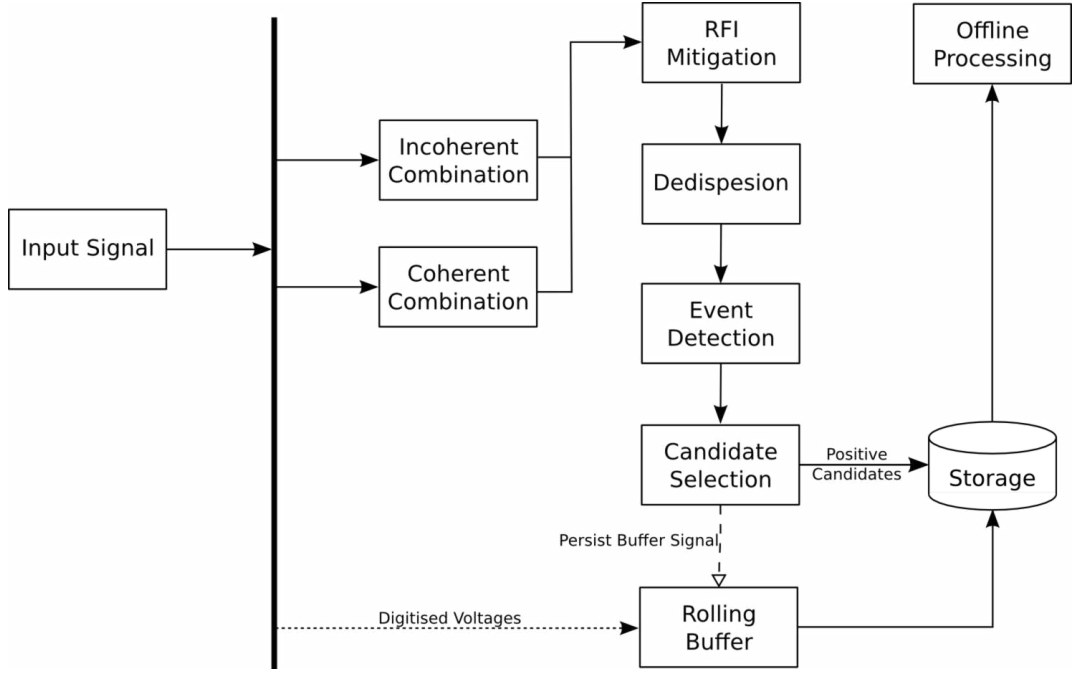


Figure 1.8: Generic fast transient detection pipeline.

step is typically performed prior to dedispersion, through clipping and thresholding algorithms, or in the case where multiple beams are being processed, during event detection with coincidence matching techniques.

(iv) *Dedispersion*: As discussed in section 1.2, astronomical signals pass through a cosmic medium of unknown dispersion measure, therefore detection needs to be trialled for many DMs. This is generally the costliest step in a single-pulse transient pipeline, computationally (for pulsar surveys, the dedispersion cost can be dwarfed by searches for binary systems). The DM range to be trialled depends on several factors, including the sky location being observed, the observation parameters, as well as the available computational resources.

(v) *Event Detection*: An event detection algorithm needs to be applied to each dedispersed time series. This generally takes the form of thresholding and matched filtering techniques, and is discussed in detail by [Cordes and McLaughlin, 2003]. This step can generate a large number of detections, especially in observations which are dominated by terrestrial RFI.

(vi) *Candidate Selection*: The large number of detections from the event detection stage need to be grouped together, such that detections caused by the same event can be processed collectively. These clusters of detections can then be classified as either caused by RFI or by an astrophysical event. Several techniques can

be employed to perform this classification, including clustering, pattern-matching and machine learning techniques, and is an area of ongoing research.

(vi) *Storage*: The rolling buffer stores a small period of data, and on receipt of a trigger from the candidate selection stage, will dump the voltage data to storage for offline processing, which could include RFI filtering information, analysis of the candidate detection stage as well as a correlation of the dish or station beams for source localisation and imaging.

1.5.1 Single Pulse Searches S/N

After correcting for dispersion, each dedispersed time series must be searched individually for pulses with amplitudes above some S/N threshold. This search is generally performed using matched filtering techniques. Given a time series of predominantly Gaussian noise of known mean and standard deviation, these techniques search for individual events that deviate by several standard deviations from the mean. Considering a rectangular pulse of amplitude S_{peak} and width W , and for the optimal case when W is equal to the sampling time t_{samp} , [Cordes and McLaughlin, 2003] show that the S/N ratio of the pulse

$$S/N = \frac{S_{\text{peak}}W}{S_{\text{sys}}} \sqrt{\frac{N_p \Delta v}{W}} \quad (1.27)$$

where S_{sys} is the system equivalent flux density (SEFD), N_p is the number of polarisations summed and Δv is the receiver bandwidth. For a fixed pulse area it follows that $S/N \propto 1/\sqrt{W}$, thus narrower pulses are easier to detect than broader ones, however, a low-amplitude, broad pulse is more easily detectable than a sharp narrow pulse if its area is sufficiently large. In general, W will not usually be a good match to t_{samp} and the S/N will be less than expected from equation 1.27. In order to match optimal detections more closely the time series is smoothed by successively adding groups of adjacent samples and searching for statistically significant events. If the true pulse shape and width are unknown, the smoothing approach is a straight-forward and efficient approximation to optimal detection.

A sensible choice for S/N threshold should be made to avoid recording too many candidate pulses that most likely are caused by random noise fluctuations. For the ideal case of Gaussian noise with zero mean and unit standard deviation, [Cordes and McLaughlin, 2003] show that the number of events expected to occur

by chance above some threshold S/N_{\min} is

$$n(> S/N_{\min}) \sim 2n_{\text{samp}} \int_{\text{SNR}_{\min}}^{\infty} \exp(-x^2) dx \quad (1.28)$$

where n_{samp} is the number of samples in the time series. Requiring that $n < 3$ usually leads to $S/N_{\min} = 4$. In practice, however, radio frequency interference (RFI) usually increases the number of false detections so that a more practical S/N threshold is 5-6 [Lorimer and Kramer, 2005].

1.5.2 Transient Surveys

Despite the scientific potential, the transient and time variable sky is a relatively unexplored region of parameter space. The restricted survey speeds of older generation radio facilities has meant that detections of radio transients lag far behind that which is possible with, for example, X- and γ -ray instruments. Four dominant methods have so far been used to detect radio transients: (i) dedicated surveys, (ii) multi-wavelength triggered detections, (iii) archival studies and (iv) serendipitous and commensal detections.

Major facilities are currently under construction, and older facilities are undergoing dramatic upgrades, driven in large part by the desire to achieve the full Square Kilometre Array (SKA) in the next decade ([Carilli and Rawlings, 2004]). [Colegate and Clarke, 2011] list some current and ongoing radio searches of the high time resolution universe, together with the event rate per beam, using the figures of merit discussed in section 1.4.1. The event rate per beam have been normalised to the incoherent combination mode of SKA₁ low band dishes, based on the specification listed in [Dewdney et al., 2010]. Their table is reproduced in table 1.1.

Several large collaborations are conducting or planning surveys for fast radio transients. This includes groups using LOFAR [Stappers et al., 2004, Fender, 2012, Hessels et al., 2008], the VLBA: V-FASTR [Wyath et al., 2011]; Parkes: High Time Resolution Universe Survey [Keith et al., 2012]; ASKAP: CRAFT [Macquart et al., 2010]; MeerKAT: TRAPUM (PIs: Stappers and Kramer), Arecibo: PALFA [Cordes et al., 2006, Deneva et al., 2009] and the GBT: GBNCC (PI: Scott Ransom). Although highly sensitive in their own right, these surveys are envisaged as precursors to those that will ultimately be conducted with the SKA [Cordes, 2009]. A crucial element to the detection of rare events is the large FoV afforded by the new technologies employed by these surveys. The Parkes telescope utilises multi-

Experiment ^a	Telescope and status	v_{centre} (MHz)	Δv (MHz)	B_{max} (km) ^b	$\mathcal{R}_{\text{beam}^{-1}}$ (normalised) ^c	Max. beams available
Archival searches ^d	Parkes	N/A	-	-	-	-
Fly's eye radio transient search ^e	ATA completed	1420	210	N/A	10^{-3} (fly)	42
HTRU Survey ^f	Parkes (operational)	1352	340	N/A	10^{-2}	13
PALFA Survey ^g	Arecibo (operational)	1440	100	N/A	10^{-2}	7
V-FASTR ^h	VLBA (operational)	1400	64	6000	10^{-2} (inc.)	1
LOFAR Transients KSP ⁱ	LOFAR (in progress)	120	32	<100	10^{-1} (inc.) 10^{-4} (coh.)	1* thousands*
CRAFT Survey ^j	ASKAP (planned)	1400	300	6	10^{-2} (inc.) 10^{-6} (coh.)	36 N/A
SKA ₁ AA-low		260	380	200	1 (inc.) 10^{-1} (coh)	hundreds* thousands*
SKA ₁ LB dishes		725	550	200	1 (inc.) 10^{-2} (coh.)	1 thousands*

Table 1.1: Radio searches of the high time resolution universe. Table reproduced and adapted from [Colegate and Clarke, 2011, table 1]

^a Only experiments within SKA1 frequencies (70 MHz - 3 GHz) are listed, and surveys which are insensitive to single pulses are excluded. N/A is not applicable or information not available

^d Maximum baseline, for event localisation and triggering

^c Order of magnitude estimation, normalised to the incoherent combination of SKA1 low band dishes. For radio telescope arrays, the calculation is for fly's eye (fly), incoherent combination (inc.) or coherent combination (coh.). A flat spectrum and no scatter broadening is assumed.

^d [Lorimer et al., 2007, Burke-Spolaor and Bailes, 2010, Keane et al., 2011]

^e [Siemion et al., 2012]

^f High Time Resolution Universe Pulsar Survey [Keith et al., 2012]

^g Pulsar Arecibo L-Band Feed Array (ALFA) Survey. [Cordes et al., 2006, Deneva et al., 2009]

^h [Wyath et al., 2011]

ⁱ Commensal Real-Time ASKAP (CRAFT) Survey. [Hessels et al., 2008, van Leeuwen and Stappers, 2010]

^j [Macquart et al., 2010]

* Limited by the available beamformer processing and data transport

beam technology, the MWA and LOFAR use aperture array technology which can, in principle, detect objects over a large fraction of the visible sky, while APERITIF and ASKAP employ focal plane aperture array technology to achieve a FoV of 8 and 30° respectively.

1.6 Thesis Outline

This thesis works towards building a real-time, GPU-based, non-imaging backend for radio telescopes, emphasising on online fast transient discovery. This poses several challenges due to the considerable amount of computational resources required and high data rates involved. We propose GPUs as ideal candidates to tackle various aspects of the several stages within this pipeline, and present a viable prototype which we demonstrate to be scalable to large-N, wide bandwidth telescopes, and propose a tentative architecture for several components of SKA₁.

Chapter 2 starts with a brief overview of many-core technologies, with special emphasis on GPUs. A generic GPU framework is then presented, which parallelises input, processing and output stages across multiple GPUs and CPUs. The challenges posed by dedispersion are then discussed, and a GPU implementation for incoherent (direct method) as well as coherent dedispersion is presented, together with performance and speedup benchmarks.

In chapter 3 we give a brief overview on the current state of the art transient detection pipelines, and present our own. The overall software architecture is first discussed, after which we describe in detail the various processing stages, including: bandpass correction, channel thresholding, spectrum thresholding, dedispersion, median filtering, detrending, candidate selection through clustering and cluster parametrisation, as well as quantisation and data persistence. We then present various figures of merit for some of these stages, particularly the classification stage. We also discuss several performance benchmarks which were performed on the pipeline, and compare this to similar state of the art pipelines.

In chapter 4 we introduce the BEST-II SKA pathfinder and then describe the digital backend deployed by a team based in Oxford, to which we attached our transient detection pipeline. Real-time additions to the pipeline are discussed, and then we move on to list several test observation data products, including observations of PSR B0329+54.

In chapter 5 we discuss the applicability of GPUs for beamforming aperture arrays and describe our multi-beam coherent beamformer. Several performance benchmarks are then presented, highlighting the major bottleneck for such implementations. Real-time benchmarks are also presented, aimed to simulate the BEST-II digital backend and highlight a use case where transient detection pipelines and beamforming kernels can be integrated, enabling dynamic observations with real-time beam observation adjustment possibilities.

Chapter 6 starts by briefly exploring the digital signal processing challenges posed by SKA₁ and propose a speculative design for station-level, GPU-based beamforming and transient detections for SKA₁-low, providing estimates for hardware requirements and cost. The same procedure is then applied to beamforming and dedispersion stages of the SKA₁-mid non-imaging pipeline.

Chapter 7 draws some conclusion on the work presented in this thesis and presents some possible extensions and future work.

CHAPTER 2

DISPERSION REMOVAL TECHNIQUES AND IMPLEMENTATION

The need for higher computational power, required by online transient detection systems, has been the driving force of several projects in recent year. Conventional systems are either composed of several interconnected servers/workstations (compute clusters) or employ custom hardware for specialised processing. Field Programmable Gate Arrays (FPGAs) have played an important role in this, especially with the development of custom programmable boards such as CASPER's FPGA-based ROACH board which reduces the amount of time required to test prototype digital designs for use in radio astronomy. Graphics Processing Units (GPUs) have gained popularity and support in recent years, and their application to radio astronomy has been extensively investigated (see [Barsdell et al., 2010] for a theoretical analysis of several algorithms which would benefit a performance gain when implemented on GPUs). Several existing instruments, including pulsar timing experiments (for example, see [Allal et al., 2009, Ransom et al., 2009]), transient detectors [Serylak et al., 2012], spectrometers [Kondo et al., 2009] and array correlators [Clarke et al., 2013a] make use of GPUs to speed up processing.

Most dispersion removal techniques are inherently parallel over multiple dimensions and so would potentially gain a performance boost when run on GPUs. In this chapter we investigate this for two dedispersion algorithms: direct (incoherent) dedispersion and coherent dedispersion. Both techniques were implemented and optimised for the latest GPU architecture developed by NVIDIA. Where applicable, performance benchmarks are also presented compared to existing implementations.

2.1 Incoherent Dispersion Removal Techniques

The effect of dispersion on a short-duration pulse is to smear it out in time. In order to determine the emitted pulse shape, or to detect the pulse with maximum S/N, the effect of dispersion must be corrected for through a process known as dedispersion. Two methods are generally employed. *Incoherent Dedispersion* works on channelised, detected data, whereupon a range of trial DMs are searched by summing across frequency channels, after delaying each channel according to the trial DM of interest. *Coherent Dedispersion* operates on raw telescope voltages and involves convolving the telescope voltages with the impulse response corresponding to the inverse of the ISM. This is generally used for pulsar monitoring, when the DM is approximately known, as the inverse filtering preserves the emitted pulse shape more faithfully than incoherent dedispersion. In this section we describe some of the incoherent dedispersion techniques which are in use, whilst in section 2.5 we describe coherent dedispersion in more detail.

2.1.1 Direct Dedispersion

The simplest way to correct for the dispersion effect is to split the incoming frequency band into a number of narrow, independent frequency channels, record the signal in each band separately and then apply an appropriate time delay to each channel, such that the received pulses arrive at the output of each channel at the same time. These delays are calculated by using the dispersion relationship (equation 1.15), where f_2 is a channel with a frequency f_{chan} and f_1 is the reference frequency f_{ref} (usually the top frequency). These delayed channel frequencies can then be summed to produce a dedispersed time series.

The direct dispersion removal technique, also known as brute-force dedispersion, sums the frequency channels along a quadratic dispersion trail for each input time sample. This process is repeated for every DM value to be processed, and thus generates a set of dedispersed and summed time series for input dataset A as follows:

$$D_{d,t} = \sum_c^{N_c} A_{f,t+\Delta t(d,c)} \quad (2.1)$$

where subscripts d , t and c represent the DM value, time sample and frequency channel respectively, N_c is the total number of frequency channels and $D_{d,t}$ represents the dedispersed trail (d,t) . $\Delta t(d,v)$ is a discretised version of equation 1.15, and gives the time delay relative to the start of the band in integer time samples

for a given DM and frequency channel:

$$\Delta T(c) \equiv \frac{k_{\text{DM}}}{\Delta t} ((v_0 + c\Delta v)^{-2} - (v_0)^{-2}) \quad (2.2)$$

$$\Delta t(d, c) \equiv \text{round}(\text{DM}(d)\Delta T(c)) \quad (2.3)$$

where Δt is the sampling time, v_0 is the frequency at the top of the band in MHz and Δv is the width of a frequency channel. $\text{DM}(d)$ represents the range of DM trials to be computed. This technique has a complexity of $\mathcal{O}(N_t N_c N_{\text{DM}})$, where N_{DM} is the total number of dispersion measure trials and N_t is the number of time samples.

When dedispersing for large DMs the dispersed signal is significantly smeared within a single frequency channel. This occurs when the gradient of the dispersion curve on the time-frequency grid is less than unity (referred to as the ‘diagonal DM’). When this effect is significant it becomes inefficient to dedisperse the time series at its full sampling rate. One option is to reduce the sampling interval by a factor of 2 when the DM exceeds the diagonal, and then repeat the procedure at two times the diagonal, four times the diagonal, and so on. We refer to this process as “time binning”, and it results in an overall reduction in computational cost, however it also introduces a degradation in pulse S/N if the intrinsic pulse width is comparable to that of the dispersion smear.

Transient surveys generally require dedispersion over a large number of DM trials, of order $\mathcal{O}(10^4)$, requiring considerable computational resources. Methods for speeding up these processes using accelerator technologies, such as GPUs, will be discussed shortly. [D’Addario, 2010, Clarke et al., 2011, Clarke et al., 2013b] describe an FPGA-based implementation of a transient detection system, based on direct dedispersion, for the CRAFT survey, and examine the computational and S/N performance of their system. In chapter 3 we’ll compare their system to our GPU-based transient detection pipeline. An alternative to speeding up implementations is to increase computational efficiency.

2.1.2 Tree Dedispersion

General brute-force dedispersion algorithms involve many redundant operations, in that they add the same sample multiple times for different DMs. The Taylor tree method [Taylor, 1974] attempts to reduce the complexity of the dedispersion computation from $\mathcal{O}(N_t N_c N_{\text{DM}})$ to $\mathcal{O}(N_t N_c \log_2(N_c))$. A Taylor tree consists of a network of delay and sum elements interconnecting N inputs to N outputs, a

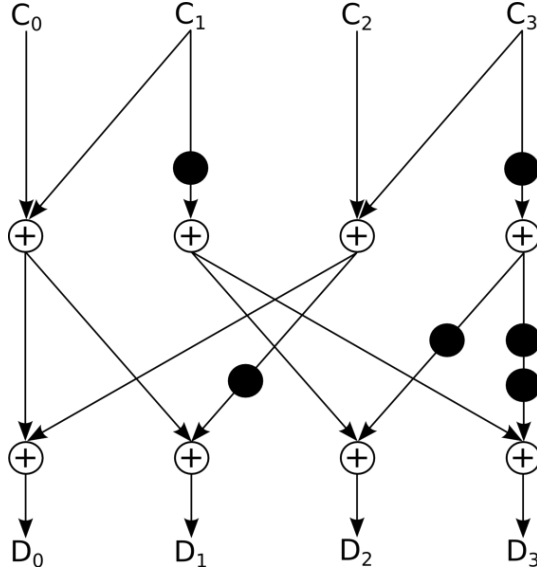


Figure 2.1: Visualisation of the tree dedispersion algorithm. N channels are input to the system, and N dedispersed time series are then generated and output. Arrows represent data flow, whilst solid circles indicate unit time delays.

4-channel version of which is shown in figure 2.1. The basic algorithm generates the dedispersed time series as follows:

$$D'_{d',t} = \sum_c^{N_c} T_{c,t+\Delta t'(d',c)} \quad (2.4)$$

$$\Delta t'(d',c) = \text{round} \left(d' \frac{c}{N_c - 1} \right) \quad (2.5)$$

where d has a value $0 \leq d < N_c$. Due to the regularisation scheme employed by the algorithm, the delay function $\Delta t'(d',c)$ becomes a linear function of c that ranges from 0 to exactly d' across the band, resulting in a DM range of

$$\text{DM}(d') = \frac{d'}{\Delta T(N_c - 1)} \quad (2.6)$$

One of the shortcomings of Taylor trees is that they implement linear approximations to dispersion, not proportional to the inverse-square of the frequency, which are less accurate for lower frequencies and wider bandwidths. Additionally, the computed dispersion measures are constrained to those given by equation 2.6, and the number of input frequency channels, N_c , must be a power of two. The last constraint is not a significant one, since zero-padded frequency channels can be inserted between existing channels, thus spreading the signal out in frequency,

with the additional effect of alleviating somewhat the first constraint. Alternative methods can be employed to work around these limitations.

Piecewise linear tree

The piecewise linear tree method [Manchester et al., 1996] approximates the dispersion curve using piecewise linear segments, where the input data is divided into N_{sub} subbands of length

$$N'_c = \frac{N_c}{N_{\text{sub}}} \quad (2.7)$$

where the n^{th} subband starts at frequency channel nN'_c . This method involves two stages of computation, the first of which involves applying the basic tree dedispersion algorithm to each subband independently

$$S_{n,d',t} = \sum_{c'}^{N'_c} T_{c_n+c',t+\Delta t'(d',c_n+c')} \quad (2.8)$$

This stage approximates the quadratic dispersion trail with a linear one. The linear DM in the n^{th} subband that approximates the true DM indexed by d is computed as follows:

$$d'_n(d) = \Delta t(d, c_{n+1}) - \Delta t(d, c_n) \quad (2.9)$$

$$= \text{round}(\text{DM}(d)[\Delta T(c_{n+1}) - \Delta T(c_n)]) \quad (2.10)$$

During the second stage, the dedispersed subbands are then combined to approximate the result of equation 2.1

$$D_{d,t} \approx \sum_n^{N_{\text{sub}}} S_{n,d'_n(d),t+\Delta t''_n(d)} \quad (2.11)$$

$$\Delta t''_n(d) = \text{round} \left(\text{DM}(d) \sum_m^n \Delta T(c_{m+1}) - \Delta T(c_m) \right) \quad (2.12)$$

It should be noted that this method introduces additional smearing into the dedispersed time series as a result of approximating the quadratic dispersion curve with a piecewise linear one. [Barsdell et al., 2012, appendix B] derive an analytic upper limit for this smearing.

Frequency-padded tree

An alternative approach is to linearise the input data by changing the frequency coordinates, whereby $\Delta T(c)$ is stretched to a linear function $\Delta T'(c') \propto c'$, such that

$$c' = \text{round} \left(\frac{v_0}{2\Delta v} \left[1 - \left(1 + \frac{\Delta v}{v_0} c \right)^{-2} \right] \right) \quad (2.13)$$

Evaluating $c = N_c$ gives the total number of frequency channels in the linearised coordinates, which determine the additional computational overhead introduced by the procedure. This number must be rounded up to the nearest power of 2 before the tree dedispersion algorithm can be applied. This linearisation is generally implemented by padding the frequency dimension with blank channels such that the real channels are spaced according to equation 2.13.

Computing Larger DMs

The basic tree dedispersion algorithm computes exactly the DMs specified by equation 2.6, however it is often necessary to search a much larger DM range. This can be achieved by transforming the input data and repeating the dedispersion computation for each transformed series. The following operations can be used to compute an arbitrary range of DMs:

- i Apply the tree algorithm, or any variant mentioned above, to obtain DMs from zero to DM_{diag} (the diagonal DM)
- ii Apply a time delay across the entire band, such that $\Delta t = c$
- iii Apply the dedispersion algorithm to obtain DMs from DM_{diag} to 2DM_{diag}
- iv Repeat from step (ii) to obtain DMs up to $2^n \text{DM}_{\text{diag}}$

An alternative approach is to downfactor, or “bin”, the input time series. This method provides a performance benefit, however at the cost of a minor reduction in the S/N for pulses of intrinsic width near the DM smearing time. This procedure is applied as follows:

- i Apply the dedispersion algorithm to obtain DMs zero to DM_{diag}
- ii Apply a time delay across the band
- iii Apply the dedispersion algorithm to obtain DMs from DM_{diag} to 2DM_{diag}

- iv Downfactor the time series by a factor of 2 by summing adjacent samples
- v Apply the dedispersion algorithm to obtain DMs from 2DM_{diag} to 4DM_{diag}
- vi Repeat from step (iv) to obtain DMs up to $2^n\text{DM}_{\text{diag}}$

2.1.3 Subband Dedispersion

“Subband” dedispersion, a technique implemented by [Ransom, 2001, Magro et al., 2011, Barsdell et al., 2012], rather than exploiting a regularisation of the dedispersion algorithm, takes a simple approximation approach, involving two steps. In the first step the set of trial DMs is approximated by a reduced set of $N_{\text{DMnom}} = N_{\text{DM}}/N'_{\text{DM}}$ ‘nominal’ DMs, each separated by N'_{DM} trial dispersion measures. The direct method is applied to subbands of N'_c channels to compute a dedispersed time series for each nominal DM and subband. In the second step the DM trials near each nominal value are computed by applying the direct algorithm to the reduced filterbank data, formed by the time series for the subbands at each nominal DM. These data have a reduced frequency resolution $N_{\text{SB}} = N_c/N'_c$ channels across the band. The two steps thus operate at reduced dispersion measure and frequency resolution respectively, resulting in an overall reduction in the computational cost, at the expense of inducing some additional smearing into the dedispersed time series. See [Barsdell et al., 2012, appendix B] for an analytical upper-bound of this error.

2.1.4 Alternative Dedispersion Methods

Alternative methods have also been investigated which minimise the computational cost for dedispersion. [Fridman, 2010] propose the use of the Hough transform to reduce the number of trials for incoherent dedispersion, which measures the slopes of the the transient’s tracks on the time-frequency plane and thus gives estimates of the required DM range. They also state that the cumulative sum method, which is occasionally used for RFI detection, can be used to provide an estimate for transient duration matching. [Bannister and Cornwell, 2011] propose two new methods, the Chirpolator and Chimageator, which exploit the observation that when a linear chirp received by one antenna is multiplied by a delayed linear chirp received at another antenna, the result is a fixed-frequency tone whose frequency is proportional to the geometric delay. [Law and Bower, 2012] introduce the bispectrum method, which is the product of visibilities around a closed-loop

of baselines of an interferometer. The bispectrum is calibration-independent, resistant to interference and computationally efficient, and can be easily integrated into correlators for real-time transient detection. Note that the latter two methods require an interferometer, and thus are unsuitable for dishes with single feeds.

2.2 Many-core Architectures

The number of transistors on a chip continues to climb with successive generations of processor technology (Moore’s Law, which is the observation that the number of transistors on integrated circuits doubles approximately every two years), however the power available to the chip is decreasing. This has led to a “power wall” and has shifted focus of computer architecture from raw performance to performance per watt. Complex cores running at high frequencies were replaced with multiple, simpler and lower power cores within a chip. Current generation multiprocessors currently offer CPUs with a modest number of cores. This concept can be extended further to the idea of many-core architectures, where a chip can contain hundreds or thousands of simple, low-frequency and low-power cores sharing on-chip resources.

This concept is not entirely new. GPU computing is the use of GPUs, together with a host system, to accelerate general-purpose scientific and engineering applications. These devices were originally designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer, intended for output to a display. The process of shading is inherently parallel, and so GPUs evolved to include multiple shader pipelines by increasing the number of cores on the device, vastly improving computation throughput. With the development of high-level APIs, including CUDA¹ and OpenCL², this raw performance became available to general-purpose computation on the GPU. In this chapter we will focus on NVIDIA GPUs, however similar arguments can be applied to GPUs from other vendors. These have gone through a number of architectural changes, all of which affect the execution behaviour of CUDA kernels in some way, including improvements in data access coalescing, introduction of cache memory, an increase in the number of cores and the way they are partitioned, block-level and thread-level instructions, and scheduling schemes, amongst others.

¹ <https://developer.nvidia.com/category/zone/cuda-zone>

² <http://www.khronos.org/opencl/>

2.2.1 GPU Architecture

The main architectural component in GPUs is a scalable array of multi-threaded Streaming Multi-Processors (SMX), each composed of a number of Scalar Processors (SP), special functions units, double precision units, memory load and store units, and a register file, amongst additional components. The decomposition of a single Kepler SMX processor is shown in figure 2.2. The multiprocessor creates, manages and executes concurrent threads in hardware with minimal scheduling overhead, supporting very fine-grained parallelism via fast barrier synchronisation and fast thread creation. This allows low granularity decomposition of problems, by assigning one thread to each data element.

The multiprocessor employs an architecture which NVIDIA call Single Instruction Multiple Thread (SIMT), where it maps each thread to one SP core, and each scalar thread executes independently with its own instruction address and register state, allowing hundreds of threads to run several different programs in parallel. The SIMT unit handles threads in groups of 32 parallel threads called warps (a half-warp is either the first or second half of the warp). Threads within a SIMT warp start together at the same address, but are free to branch and execute independently. Every instruction issue time, the SIMT unit selects a ready warp to execute the next instruction. The warp executes one common instruction at a time, therefore execution becomes fully efficient when all the threads within the warp are on the same instruction address. If threads undertake a data-dependent conditional branch, each branch path will be executed serially, with threads which are not on the execution path being disabled until their path is processed, or the branches converge back to the same execution path. Branch divergence occurs only within warps, different warps can execute independently. The ratio of the number of active warps to the maximum number of processable warps is called the occupancy of the GPU when running the specific kernel.

Each thread has access to several memory types: (i) one set of local 32-bit registers, (ii) a global read-write memory space accessible from all the processing units in a GPU, (iii) a parallel data cache which is shared by all SP cores, where shared memory resides, (iv) a fast, read-only constant memory space accessible from all SP cores and (v) a read-only texture cache which is shared by all SP cores. A SMX's register and shared memory are split among all the threads of the batch of blocks, so the more resources the threads use, the less blocks which can be processed at one go. A multiprocessor can execute as many as eight thread blocks concurrently. The Fermi GPU series introduced an L2 cache which caches



Figure 2.2: NVIDIA GK110 SMX Block Diagram: 192 single precision CUDA cores, 64 double precision units, 32 special function units (SFU), and 32 load/store units (LD/ST). Reproduced from [NVIDIA, Corporation, 2012]

global memory data, and is global to all the SP cores.

CUDA is a general purpose computing architecture that leverages the parallel compute engine in NVIDIA GPUs to solve complex and demanding computational problems. It comes with a software environment which allows the use of C as a high-level programming language. CUDA has three key abstractions: a hierarchy of thread groups, shared memories and barrier synchronisation, which are exposed to the programmer as simple language extensions. They provide fine-grained data and thread parallelism, guiding the programmer to partition the problem into coarse sub-problems which can be solved by parallel threads within the block.

	NVIDIA K20	Xeon Phi SE10¹	Xeon E5-2670¹
Single PTP	3.52 TF/s	2.15 TF/s	332.3 GF/s
Double PTP	1.17 TF/s	1.07 TF/s	161.4 GF/s
Cores	2688 (CUDA)	61	8 (16 HT ²)
Clock Speed	732 MHz	1.1 GHz	3.8 GHz ³
Memory	5 GB GDDR5	8 GB GDDR5	≤750GB (DDR3) ⁴
Memory BW	208 GB/s	352 GB/s	102.4 GB/s
TDP ⁵	235 W	300 W	135 W

Table 2.1: Specification comparison between the NVIDIA K20, Intel Xeon Phi SE10, Intel Xeon E5-2670. Peak Theoretical Performance (PTP) is calculated as follows: $F \times N_{\text{cores}} \times C$, where F is the number of FLOPS per clock cycle and C is the clock speed in Hz.

¹ Devices developed by Intel

² Hyper-Threaded Cores

³ With Intel Turbo Boost enabled

⁴ RAM DIMMS attached to host motherboard which must support similar specifications

⁵ Thermal Design Power

Each block can be scheduled on any processor core so that a compiled CUDA program can execute on any number of of processor cores. Thread blocks are in turn grouped as a grid, which is a high level abstraction of the problem at hand, with the thread blocks representing a coarse partitioning of the problem, and threads in turn represent fine-grained decomposition.

2.2.2 State of the Art Many-core Devices

The latest Kepler-based GPUs, the architecture of which is depicted in figure 2.2, have a single-precision peak theoretical performance of around 3.5 TFLOP/s, vastly exceeding that of current high-end CPUs (order of 330 GFLOP/s). Table 2.1 lists some specifications for the NVIDIA K20, Intel Xeon Phi SE10 and Intel Xeon E5-2670. The Intel Xeon Phi is a new many-core PCI device developed by Intel having up to 60 cache-coherent Xeon cores connected in a ring buffer around 8 GB of GDDR5 RAM with a high memory bandwidth. The most attractive feature of this device is the ease of deployment of existing threaded and MPI applications due to their support for the x86-based instruction set. Bandwidth-limited algorithms will also benefit from the larger cache and higher memory bandwidth. To date there has been no published investigation into the suitability of this device for radio astronomy signal processing. It should be noted that the peak theoretical performance of CPUs as well as the Intel Xeon Phi, can only be achieved through

the use of vector registers (256-bit AVX³ in the case of recent Xeon CPUs and 512-bit AVX for the Xeon Phi). Recent compilers can automatically vectorise data parallel implementations to make use of these features, however in order to achieve optimised high performance low-level intrinsics/assembly code is required.

2.2.3 Mapping algorithms to Many-core Architectures

Many-core architectures exhibit a number of characteristics that can impact the performance of an algorithm. To fully utilise these massively-parallel architectures, algorithms must exhibit a high level of parallel granularity, such that the same operations can be performed on a number of data items in parallel. Algorithms which satisfy this requirement are considered to be data parallel. Additionally, these devices generally have high internal memory bandwidths and high latency (memory transfer time) costs can limit the achievable performance of an implementation if the data access pattern does not reflect the underlying memory mechanism of the device. These can achieve a high memory throughput if data locality (neighbouring threads access memory elements which are physically close in memory) is maximised. In CUDA-terms, when threads in a warp access contiguous memory locations, each individual request will be coalesced into a single request, thus reducing the overall bandwidth required.

The cores in many-core devices should be kept busy as much as possible, so algorithms should have a high arithmetic intensity, the lack of which would result in a low compute to memory access ratio. Such algorithms are referred to as bandwidth-limited, such that their performance is limited by the rate at which memory can be accessed rather than the arithmetic instructions which can be processed. Fermi and Kepler GPUs have an inbuilt block-level cache and latency hiding schemes which can help, however for algorithms with large strided or irregular access patterns it is usually beneficial to implement a custom caching scheme using shared memory as this will avoid loading extra data values within cache lines. These are limited resources, so care should be taken in designing optimal data partitioning schemes.

For compute-limited algorithms a high level of Instruction-Level Parallelism (ILP) is required to achieve high intensity computations, coupled with the avoidance, or limited use of, high latency calls such as atomic operators, modulus and division, and math library calls. ILP is a measure of how many independent oper-

³ Advanced Vector Extensions: <http://software.intel.com/en-us/avx>

ations can be performed simultaneously by a single thread, and depends highly on the hardware design of the underlying architecture, generally requiring multiple pipelines to overlap instructions. ILP is essential to achieve the peak theoretical performance of Kepler GPUs.

GPUs, and performance accelerators in general, are connected to the host motherboard via interfacing links, with PCI-express currently being the interface of choice. The data on which the GPU operates has to reside in GPU memory⁴, therefore a CPU to GPU transfer has to be issued, and an additional transfer is required to copy the results back to host memory for further processing. The transfer rates are limited by the interface bandwidth, which for a 16x-lane PCIe-3 link is about 15.75 GB/s (bi-directional). This can pose a bottleneck for high throughput systems, or for algorithms with a low arithmetic intensity, and can limit the applicability of GPU to certain scenarios, some of which will be encountered in chapter 6.

2.3 GPU Framework

GPU kernels require data transfers to and from host memory, as well as some interfacing mechanism to synchronise GPU execution with data acquisition, data transfer and host execution. This overlapping overhead must be minimised in order to reduce the run time of the entire application. A pipelining mechanism was developed which overlaps data acquisition, GPU execution and post-processing of dedispersed data, as the schematic in figure 2.3 shows. This pipeline consists of three main stages:

Data Acquisition: GPU kernels require data to process, which can originate from different locations, such as files and real-time network streams. These sources can have different latencies, and so data acquisition needs to be performed in parallel so as not to interfere with kernel execution, especially when the pipeline is run in real-time mode. An interface is provided which accepts large data buffers that are copied directly to GPU memory for processing. These buffers are populated by custom data interpreters. Additional operations performed by this thread include: parsing and initialisation of input parameters, threads and synchronisation objects, as well as error handling.

⁴ This is not technically necessary with the latest version of CUDA, however in principle data still has to be transferred between the two memory spaces.

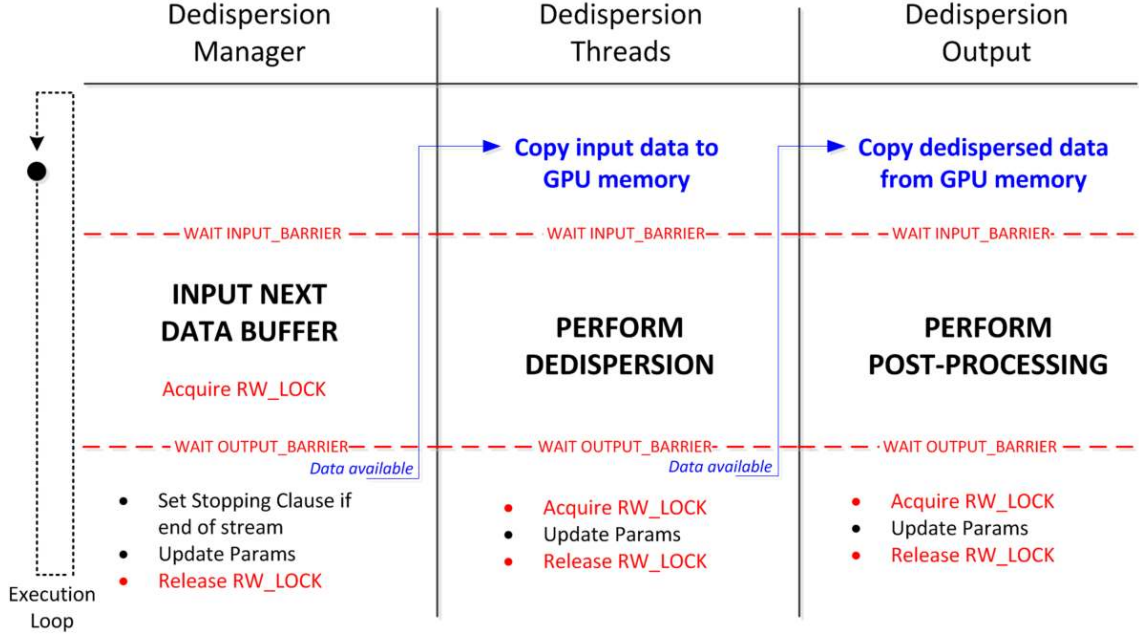


Figure 2.3: Each thread has three main stages: the input, processing and output stages. Data has to flow from one thread type to the next, so synchronisation objects have to be used to make sure that no data is overwritten or re-processed. Barriers and RW locks are used to control access to critical sections.

Kernel Execution: Execution parameters are parallelised across multiple GPUs so as to meet realtime requirements. A CPU thread is created for each execution instance and is mapped to a single GPU. Multiple CPU threads can be mapped to the same GPU. Each thread is responsible for copying its section of the input buffer to GPU memory, launching the execution kernel and copying the resultant output to host memory.

Post Processing: A single host thread is responsible for post-processing, and where required OpenMP threads are used to speed up compute-intensive operations.

Currently this framework assumes an homogeneous system (with only one type of GPU device attached to it), and does not perform any load-balancing between the devices. The host threads are split into three conceptual processing stages, which are guarded by several thread-synchronisation mechanisms. The three stages are: (i) the input section, where the thread inputs data to be processed (by itself, or to send to other threads for processing) (ii) the processing section, which is the main section in the thread and the part which takes the longest to complete and (iii) the output section, where the processed buffer is output and made available to the next thread, and any parameter updates are performed. The

input and output sections are combined, so no synchronisation is required between them.

This simple pipelining framework was extended further to include additional capabilities, such as “Transient Buffer Board” mode, buffer persister, multi-beam mode and beamforming mode, which are discussed in detail in chapters 3 and 5. All performance benchmarks presented in this chapter do not include pipeline overheads, however these are generally minimal and negligible when compared to dedispersion runtime.

2.4 Direct Dedispersion GPU Implementation

An analysis for suitability of this algorithm to many-core architecture determines that:

- the algorithm is best parallelised over the “embarrassingly parallel” DM and time dimensions, with the sum of frequencies being performed sequentially
- the algorithm has a very high theoretical arithmetic intensity (same order of magnitude as the number of DMs computed)
- the memory access patterns generally exhibit reasonable arithmetic locality, however its non-trivial nature make it difficult to achieve a high arithmetic intensity

This technique has been investigated and implemented on GPUs by several authors [Magro et al., 2011, Armour et al., 2012, Barsdell et al., 2012], all of whom report a speedup of at least an order of magnitude when compared to optimised CPU-based implementations. In the following sections we’ll describe our newest implementation of this method and compare it to the implementation by [Armour et al., 2012], the major differences being the use of shared memory by the former, and cache memory by the latter to minimise data read accesses to global memory.

The key factor determining the performance of a GPU implementation of direct dedispersion is the memory access pattern used to read data from global memory, together with the structures used to keep data local to the thread in order to maximise data reuse. In a naïve implementation, when dedispersing a set of adjacent time samples, representing the data elements required to process for a DM range N_{DM} , several cells in the input space (f, t) need be to read multiple times. This behaviour was simulated by generating an empty N_t by N_f grid and a

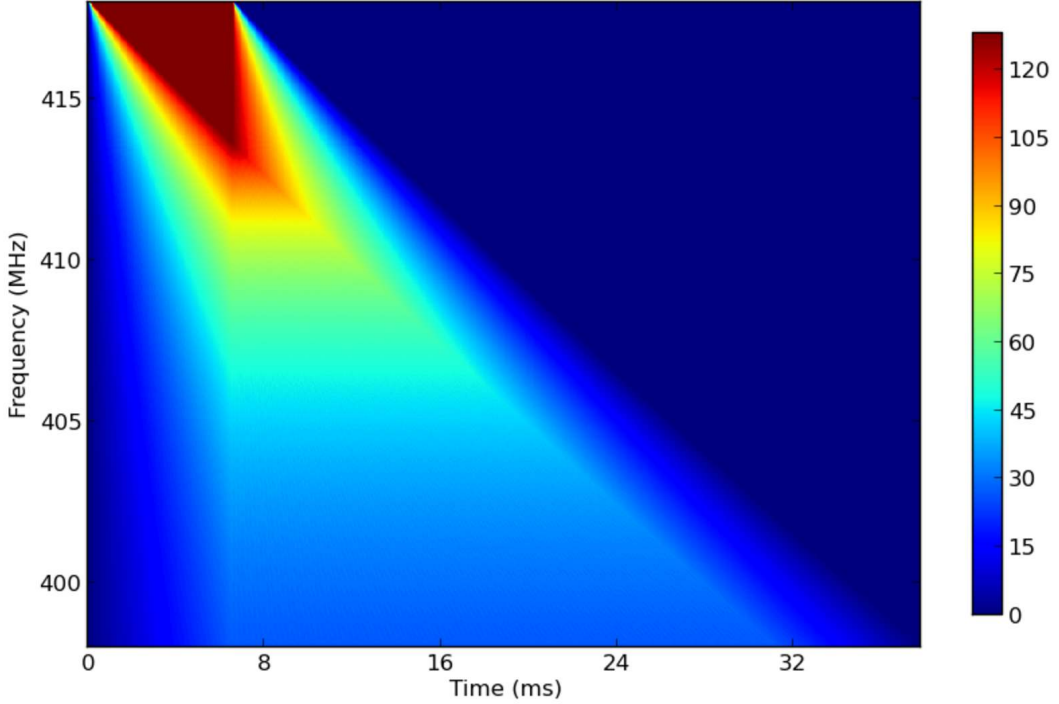


Figure 2.4: Memory access pattern simulation for a 20 MHz band, centred at 408 MHz and channelised into 1024 channels. A group of 128 adjacent time samples were traversed along the dispersion trail for 128 DM values (maximum DM of $12.8 \text{ pc}^3 \text{ cm}^{-3}$) and when a cell in (f, t) space was reached its counter was incremented.

trail for each DM value was traversed for a number of time samples. The resultant grid is shown in figure 2.4. Without any optimisations global memory would need to be accessed $N_{\text{DM}} \times N_f$ times for every dispersion measure. It is clear that this performance penalty can be alleviated by taking advantage of this behaviour.

Consider a set of adjacent time samples S being processed for a subset of the DM range D . When visiting a frequency channel c , the required number of data elements C can be calculated by taking the shift from the highest DM value D_n for the last sample S_n and subtracting the shift required by the lowest DM value D_0 for the first sample S_0 , as follows:

$$C_n = \Delta t(S_n, D_n, c) - \Delta t(S_0, D_0, c) \quad (2.14)$$

where $C_n \geq S$. C_n can be thought of as the vector of values required to process S for D , which we will refer to as the *channel vector*. Figure 2.5 provides a visual illustration of this behaviour, where the channel vector for every frequency channel is highlighted. Green cells are required by all of the samples in S , while cells in red

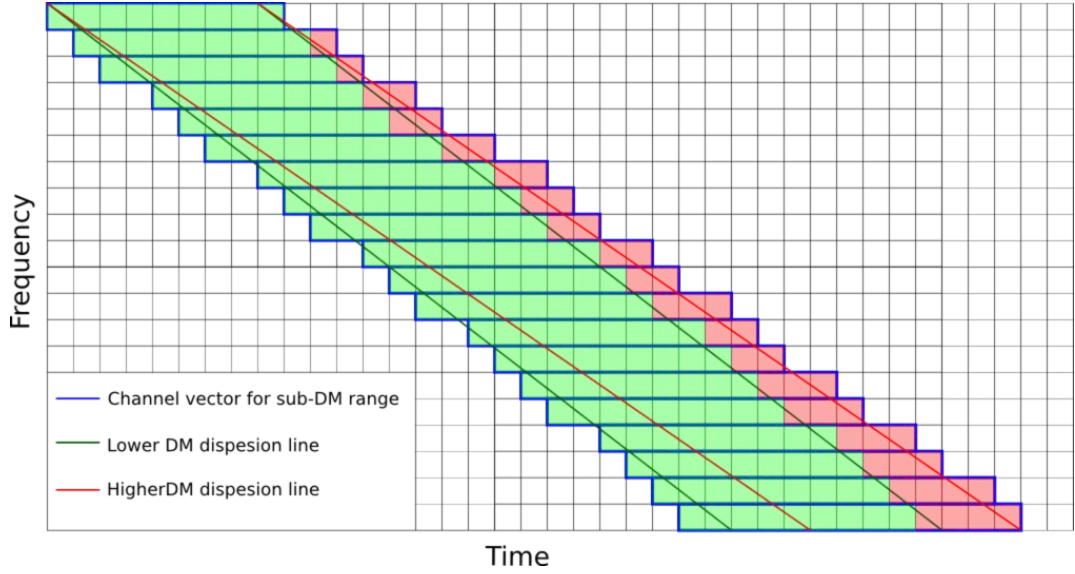


Figure 2.5: GPU memory access optimisation for direct dedispersion. The channel vectors required to dedisperse a subset of the time samples, for a subset of DM space, is loaded once to fast memory. Cells in green represent data values which are needed by all the time samples being processed, while cells in read are only required by a subset of these.

are only needed by a subset of the time samples being processed. These vectors can be loaded only once from global memory and stored in fast memory whereby they can then be read for accumulation with lower latency.

Following this behaviour, a CUDA thread block needs to process a subset of the time samples for a subrange of DM space, meaning that the CUDA grid will be a direct mapping to the output space (DM, t). Each thread within a thread block is associated with one input time sample and a number of DM trials. Local accumulators, stored in registers, one per DM value, are updated once for every frequency channel. Threads within the same thread block cooperate to load data from the input buffer into shared memory.

Details of our implementation are described in algorithm 1. Two shared memory buffers are declared, one for storing the channel vector and another for storing the sample shifts for each channel and DM combination. These shifts are required to be located in fast shared memory due to repeated access in the inner-most loop. Accumulators are stored in registers for fastest possible access (care needs to be taken not to define a large DM subrange for each thread block, otherwise these accumulators will be spilled to local memory, which is much slower). Then, for every frequency channel:

Algorithm 1 Direct dedispersion shared-memory implementation

Require: input, output, dm_delays, nchans, nsamp, tdms, maxshift
 Declare shared *accumulators*[*dedisp_threads*], *vector* and *delays*[*dedisp_dms*]

```

for  $c = 0$  to  $nchans$  do
    synchronise threads
     $inshift \leftarrow dm\_delays[block\_shift]$ 

    if  $threadIdx.x < dedisp\_dms$  then
         $delays[threadIdx.x] \leftarrow dm\_delays[block\_shift + threadIdx.x] - inshift$ 
    end if
    synchronise threads

    for  $s = threadIdx.x$  to  $s < blockDim.x + delays[dedisp\_dms - 1]$  do
         $vector[s] \leftarrow input[shift + inshift + s];$ 
    end for
    synchronise threads

    for  $d = 0 \rightarrow dedisp\_threads$  do
         $accumulators[d] += vector[threadIdx.x + delays[d]]$ 
    end for
end for

for  $d = 0$  to  $dedisp\_dms$  do
     $output[output\_shift] \leftarrow accumulators[d]$ 
end for
    
```

1. The required shifts are cooperatively loaded by all the threads within a thread block and stored in shared memory. A block-wide barrier is required to synchronise all the threads and ensure that all the shifts are loaded before any computation is performed.
2. The channel vector is cooperatively loaded to shared memory. The input buffer is stored in channel order, with time changing the fastest (and thus assumes that the data has been transposed beforehand), which means that threads within a warp will access this buffer in a coalesced manner. An extra transfer request might be issued by the GPU due to misaligned accesses caused by the shift induced by the first sample. A block-wide barrier is required to ensure all input data is available to all threads for processing.
3. For each DM value, the required shifted value from the channel vector is added to the accumulator associated with it

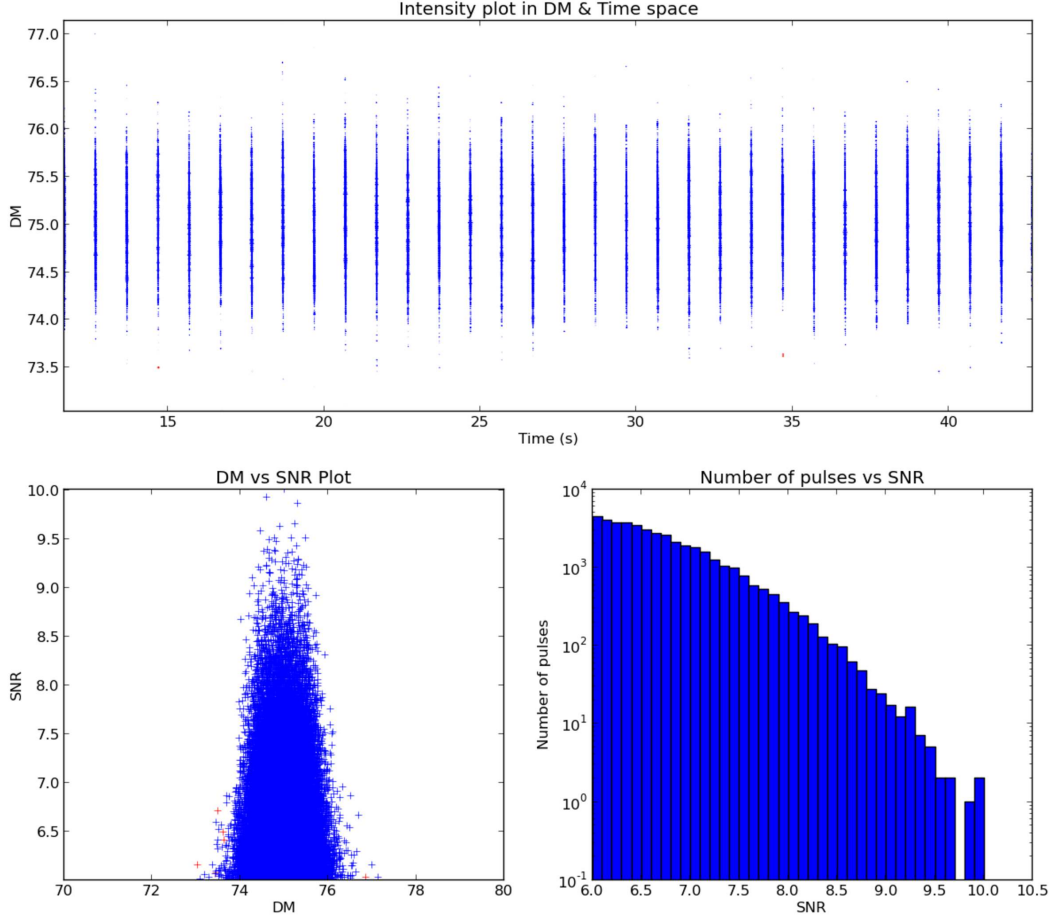


Figure 2.6: Direct dedispersion output for an input filterbank file containing a simulated pulsar. Plots show DM vs Time, weighted by the detection S/N (top), S/N vs DM (bottom left) and number of pulses vs S/N (bottom right).

Once all the frequency channels have been processed, the dedispersed and summed values are written to the output buffer in global memory.

To test the functionality of the implementation, a simulated data file centred at 153 MHz, with 6 MHz bandwidth, channelised into 1024 frequency channels, containing a pulsed signal with period 1s, duty cycle 1% and dispersed at a DM of 75 pc cm^{-3} was generated. The pulses were modelled as a top-hat pulse of height $8/\sqrt{1024} = 0.25$, which were embedded in random Gaussian noise with mean 0 and standard deviation 1. The S/N of the average simulated pulse, integrated over the frequency band, has a mean value of 8. Direct dedispersion over 4096 DM trials and a DM step of 0.02 pc cm^{-3} was performed. Figure 2.6 shows the output of the dedispersion code, which captures all detections with S/N greater than 6.

2.4.1 Performance and Benchmarks

Greater emphasis was put on making the dedispersion kernel optimised for Kepler GPUs, which makes the implementation more future proof and capable of fully utilising current top-range, high performance GPUs. These GPUs have a higher number of registers allocated per thread block, however the core clock speed is lower than in Fermi GPUs. This means that more data can be kept in fast memory, however care must be taken to better hide access latency and make sure that a high level of instruction-level parallelism is achieved. Our implementation requires two configuration parameters to be defined, and consequently optimised upon:

Number of accumulators defines how many DM values each thread will process. These are stored in registers, and thus are limited by the number of registers available to a thread block. A lower number of accumulators will decrease channel vector reuse and increase global memory bandwidth requirements, while a high number will reduce the occupancy on the GPU, and in the worst case can be spilled to local memory, significantly reducing performance.

Threads per block determines the number of time samples a single thread block will process (one per thread). A low number will reduce the number of active warps in a SMX, leading to diminished parallelism, while a higher number will increase shared memory and register requirements, reducing occupancy.

In order to find the optimal configuration for the above parameters a series of benchmark tests were conducted on an NVIDIA GTX 670 card for a wide range of parameters combinations. A 6.7 s simulated dataset centred at 408 MHz, with a bandwidth of 20 MHz, split into 1024 frequency channels, was generated and dedispersed over 4096 DM trials with a maximum DM of 409.6 pc cm^{-3} . The results of these tests are depicted in figure 2.7, which shows that increasing the number of threads per block and number of local accumulators yields a performance benefit until a global maximum is reached, after which performance starts to degrade. On the tested GPU, the optimal combination was 32 accumulators and 128 threads per block. These values were then used to benchmark other aspects of the implementation.

The scalability performance of the algorithm was also analysed and directly compared with the cache-optimised implementation of [Armour et al., 2012]. The cache-optimised implementation uses the same “channel vector” concept, however

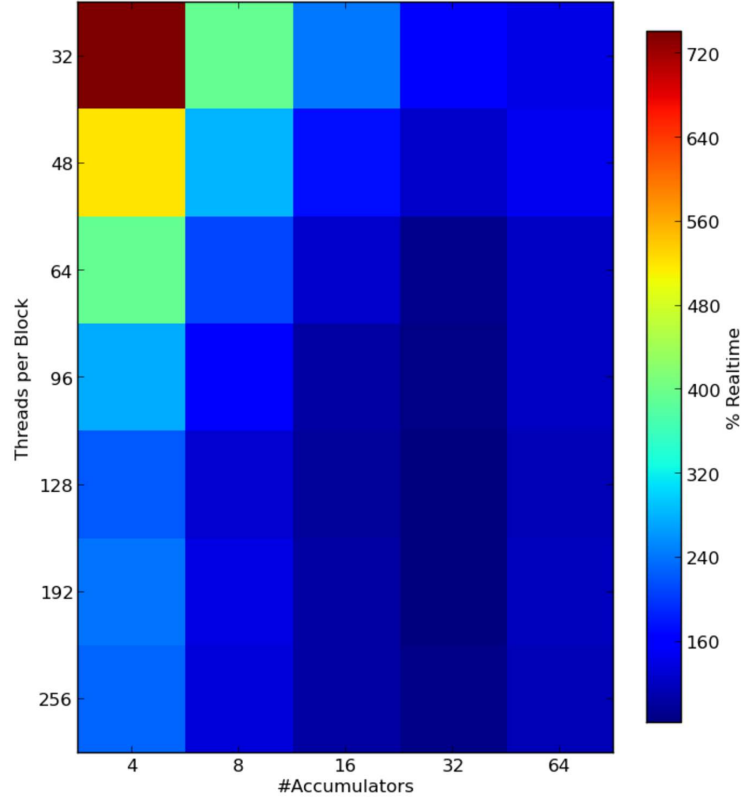
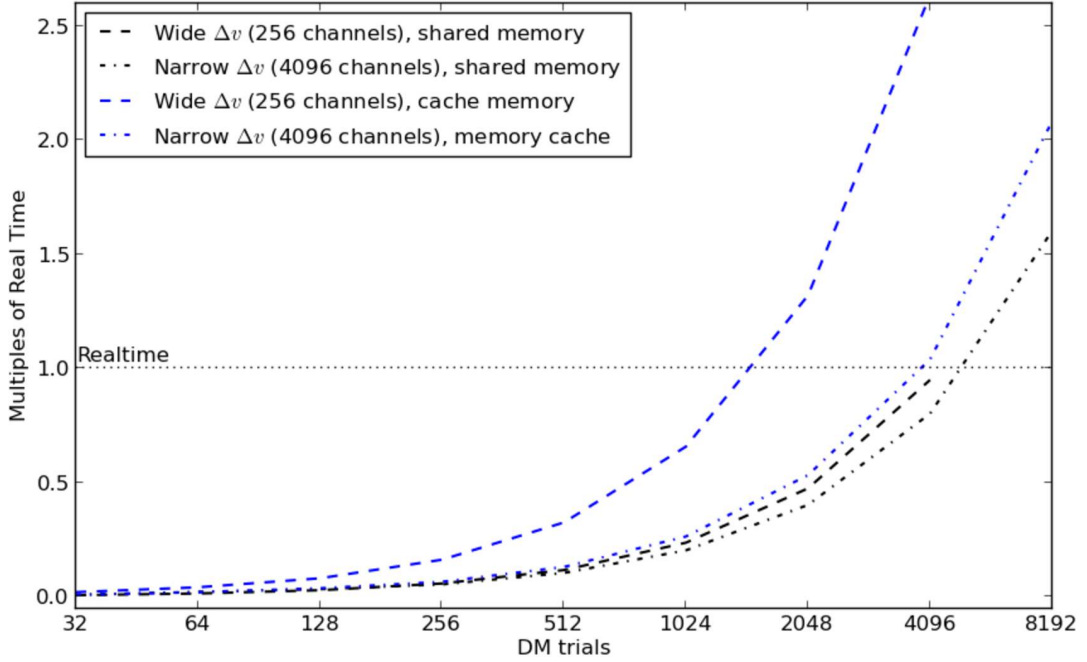


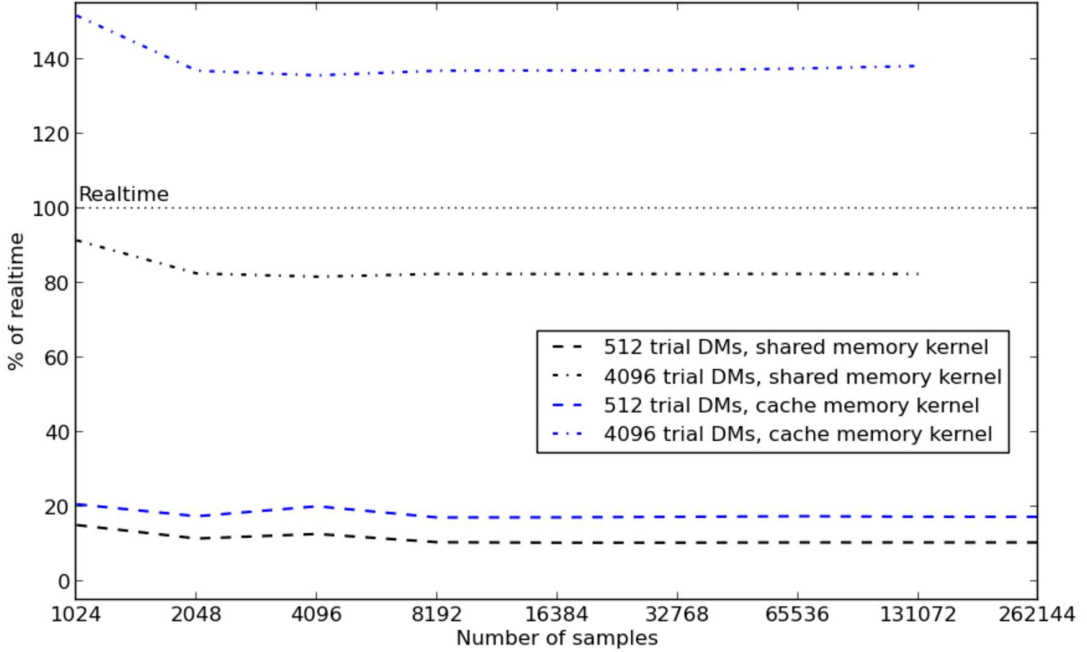
Figure 2.7: Configuration optimisation for direct dedispersion. On the tested GPU, the optimal combination was 32 accumulators and 128 threads per block.

it is assumed that this will be performed by the L1 caching mechanism introduced in Fermi GPUs. This was an improvement over the previous work on which it was based ([Magro et al., 2011]), however here we showcase the fact that implementing the caching in the kernel itself, using shared memory, provides a performance benefit. The benchmark results are presented in figure 2.8, where scaling performance for varying number of (a) DM trials and (b) time samples in the buffer are shown. Both implementations scale linearly over both varying dimensions, with the cache-optimised version showing a performance degradation for lower number of channels (wider channel bandwidth). For plot (a) a simulated dataset centred at 408 MHz, with 20 MHz bandwidth, was split into 256 (wide Δv) and 4096 (narrow Δv) frequency channels, and 1.6 s worth of data were buffered to the GPU, where both implementations were executed for a varying number of DM trials. For plot (b) the same observation parameters were used and a varying number of samples were buffered to the GPU.

The channel width-dependent performance degradation affecting the cache-optimised implementation can be attributed to the shift-dependent performance



(a) Performance for varying number of DM trials



(b) Performance for varying number of time samples

Figure 2.8: Direct dedispersion performance benchmarks for varying (a) number of DM trials and (b) number of time samples in buffer. Our implementation was compared with the one by [Armour et al., 2012]. Both implementations scale linearly over both varying dimensions, with the cache-optimised version showing a performance degradation for lower number of channels (wider channels bandwidth).

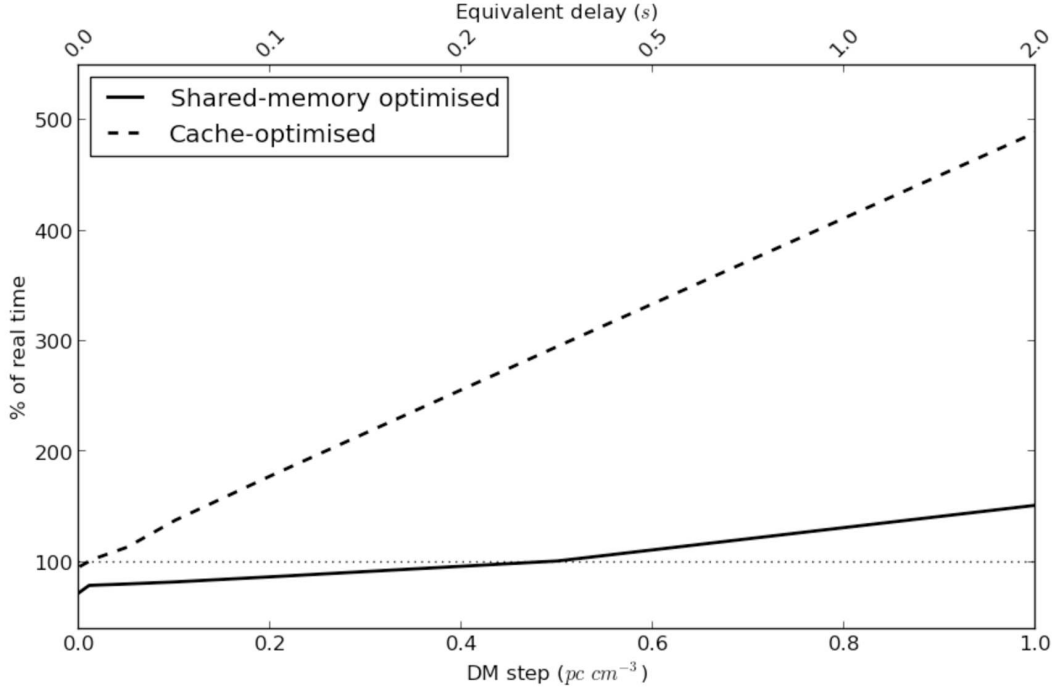


Figure 2.9: This plot shows the performance degradation of both implementations when the DM step is increased, leading to higher shifts between consecutive frequency channels for higher DM values. This results in a higher rate of cache misses for the cache-based implementation and a higher amount of shared memory usage for the shared memory based implementation.

degradation of both implementations, as depicted in figure 2.9, where a 6.7 s simulated dataset centred at 408 MHz with a bandwidth of 20 MHz, split into 1024 frequency channels, was generated and dedispersed over 4096 DM trials with varying DM steps. The equivalent delay in s refers to the delay for a DM of 1 pc cm^{-3} . Increasing the DM step generates larger shift values between consecutive channels, which could result in "skipped samples" within the channel vector. This effect is much more severe for the cache-optimised implementation since the GPU will load an entire cache line for every data value accessed, and wider shifts would result in a higher frequency of cache misses. Our implementation counters this by increasing the shared memory buffer, which results in a much lower degradation gradient. It should be noted that in real-world transient surveys this effect would be minimal since consecutive time samples are usually combined for better performance and high time resolution is not required due to excessive signal smearing within the band.

It is clear from these benchmark tests that the key to performance is data movement within the GPU. Although GPUs have very high compute capabilities, data transfers are struggling to keep the SMXs busy. On the GTX 670 a performance of around 600 Gops/s is achieved, roughly evenly split between single precision floating point, integer and shared memory access. This amounts to about 25 - 30% of peak compute capability and communication. Whilst this is a noticeable improvement from previous implementations, higher bandwidth and lower latency features in GPUs would be ideal. The Intel Xeon Phi might prove to be a more suitable platform, however the direct dedispersion kernel has not yet been benchmarked on this device.

2.5 Coherent Dedispersion

Incoherent dedispersion is limited in two ways. First, the goal of recording power vs. time make sense only on a timescale greater than

$$dt_1 = \frac{1}{dv} \quad (2.15)$$

where dv is the width of a frequency channel. This is due to time-frequency uncertainty. Secondly, it is limited by the width of the individual frequency channels, which inherently detain a small dispersion delay $dt(v)$:

$$dt(v) = 2(\mathcal{D} \cdot DM) \frac{dv}{v^3} \quad (2.16)$$

Since the signal in each frequency channel has a differential dispersion delay of $dt(v)$, incoherent dedispersion cannot localize a pulse better than this (for more details see [von Korff, 2010]). Coherent dedispersion is an alternative technique which allows better time resolution by performing the mathematical inverse of the ISM's dispersion operation. After measuring the complex voltage $v(t)$ coherent dedispersion recovers the intrinsic complex voltage as it originated from the source $v_{\text{int}}(t)$. This is then transformed into a real signal which is not affected by the dispersive effect of the ISM.

Coherent dedispersion relies on the fact that the modification of the signal can be described as the work of a 'phase-only' filter, or transfer function. In the frequency domain, for a signal centred at f_0 and bandwidth Δf :

$$V(f_0 + f) = V_{\text{int}}(f_0 + \Delta f) H(f_0 + \Delta f) \quad (2.17)$$

where $V(f)$ and $V_{\text{int}}(f)$ are the corresponding Fourier transforms of the raw voltages $v(t)$ and $v_{\text{int}}(t)$, which are non-zero for $|f| > \Delta f/2$. The delay in the ISM depends on the frequency and path length travelled, $\Delta\phi = -k(f_0 + f)d$, where $k(f)$ is the wavenumber and d is the distance to the pulsar. From this it follows that the transfer functions becomes

$$H(f_0 + f) = e^{-ik(f_0 + f)d} \quad (2.18)$$

$$= e^{+i\frac{2\pi\mathcal{D}}{(f+f_0)f_0^2}\text{DM}f^2} \quad (2.19)$$

This transfer function is determined for a given pulsar and its inverse applied to the measured, Fourier-transformed voltage. The result is then transformed back into the time domain to obtain the desired dedispersed signal. See [Lorimer and Kramer, 2005, section 5.3] for an in-depth discussion of coherent dedispersion.

A Fourier Transform can be performed using the Fast Fourier Transform (FFT) algorithm, which has a time complexity of $\mathcal{O}(N\log N)$ where N is the number of input samples. This process has to be applied separately to each DM trial and all frequency channels (in cases where the observing band is very wide and thus must be channelised into a number of narrower subbands), leading to a time complexity of $\mathcal{O}(N_f N_{\text{DM}} N \log N)$, suggesting that this is a slower technique when compared to incoherent dedispersion. However, coherent dedispersion can test for more DM trials due to higher time resolution, and thus makes it suitable for such cases. Also, it is the algorithm of choice for pulsar timing observations.

2.5.1 GPU Implementation

Several authors have investigated the applicability of the coherent dedispersion process to GPU processing [Allal et al., 2009, van Straten and Bailes, 2011]. Here we describe a simple implementation based on the overlap-save method, which is an efficient way of evaluating the discrete convolution between a very long signal (the input complex voltages in this case) and an FIR filter (the chirp function). Figure 2.10 provides a visual schematic of this technique. [van Straten and Bailes, 2011] also use this scheme for performing coherent dedispersion within their software package, and present several benchmarks of their implementation. Due to the similarity of both implementations, these tests were not replicated. Here we simply outline the algorithmic design of our implementation.

The multiplication of the chirp function with the voltage stream in the

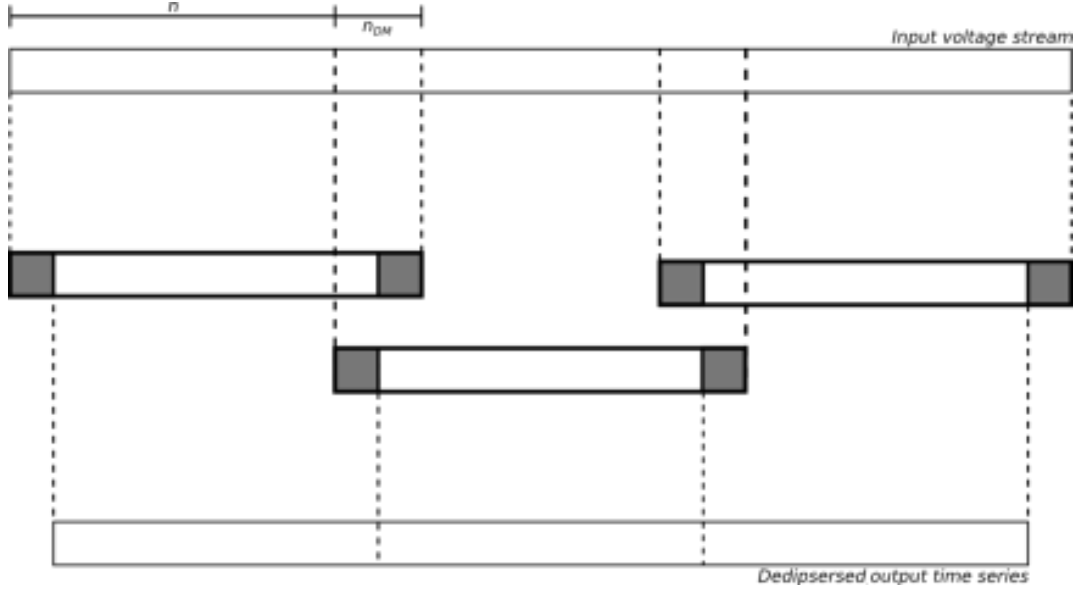


Figure 2.10: Schematic showing the overlap-save method, which is an efficient way of evaluating the discrete convolution between a very long signal, the input complex voltages in this case, and an FIR filter, here represented by the chirp function.

Fourier domain corresponds to a convolution in the time domain. The length of the voltage data for dedispersion needs to have a length of at least the dispersion delay between the upper and lower edges of the bandpass $dt(v)$ (equation 2.16). For a bandwidth dv which is Nyquist sampled, this amounts to $n_{DM} = dt(v) \cdot dv$, where n_{DM} represents the fact that this value is DM-dependent. A discrete convolution of each point of a time series of length n depends on $n/2$ points at both ends of the buffer, requiring the input voltage series to be padded. Thus the shortest data set which can be coherently dedispersed must be at least $2n_{DM}$ samples long. Long data sets are generally chosen to increase efficiency and performance. For GPU implementations this will be generally limited by the amount of memory available on the device and the performance of the FFT library being used. The 'wings' of length $n_{DM}/2$ at the beginning and end of each data set need to be ignored after convolution.

In order to minimise GPU execution time, an optimal FFT size needs to be determined for a given dedispersion length. This value is then used to fill up GPU memory using multiple overlapping blocks as per the overlap-save method. The mapping between the input CPU buffer (left) and working GPU buffer (right) is defined below:

$$N_c \cdot \left(\left(N_f - \frac{n_{DM}}{2} \right) \cdot N_B + \frac{n_{DM}}{2} \right) \Rightarrow N_c \cdot N_f \cdot N_B \quad (2.20)$$

where N_c is the number of frequency channels, N_f is the FFT size and N_B is the number of FFT blocks to be processed in parallel. Our GPU implementation consists of several processing steps:

1. A buffer containing the (possibly coarsely channelised) complex voltages is copied to GPU memory. Each frequency channel is copied separately whilst expanding each N_f set to overlap FFT execution.
2. $N_c \cdot N_B$ N_f -point forward FFTs are performed in parallel, one for each channel-block pair, using the cuFFT⁵ CUDA library. The FFT plan is created once during initialisation.
3. The coherent dedispersion kernel is launched for execution, which calculates the discretely sampled chirp function on the fly and multiplies the coefficients with the Fourier components. The kernel configuration consists of a 2D grid, with Fourier components changing in the x-direction and frequency subband in the y-direction. Each thread is responsible for the same Fourier component across all blocks. With this scheme, the chirp coefficient for the frequency bin can be calculated once and stored in registers.
4. The result is inverse Fourier transformed in place, resulting in $N_c \cdot N_B$ N_f -point buffers with $n_{DM}/2$ extra points at both edges.
5. The dedispersed buffer is copied back to CPU memory whilst simultaneously collapsing the overlap, thus removing the invalid points at the FFT edges.

The computational bottlenecks in this implementation are the Fourier transform steps. The actual computational performance will depend on the library and implementation used. Several cuFFT benchmarks are readily available. These can be used to calculate the optimal FFT size, which depends on the dispersive smearing length within the lowest frequency channel, where this smearing is widest. Data movement to and from GPU memory can also pose a considerable execution bottleneck. For this reason, when deployed on a real-time system, data expansion should be performed on the GPU itself to reduce the input data rate. Depending on the observation requirements, the output data buffer can either be folded, which is generally the case for pulsar timing, or encoded to a lower bit rate.

⁵ <https://developer.nvidia.com/cufft>

2.6 Conclusion

Online fast transient detection pipelines need to dedisperse the incoming data stream in real time, generally for $\mathcal{O}(10^4)$ DM trials when performing a blind search. For this reason, fast implementations of dispersion removal techniques have been studied and proposed in recent years. In this chapter we demonstrate the applicability of GPUs for such systems. We implement the direct dedispersion method and perform a performance benchmark and scalability study for typical usage parameters. This implementation forms part of the GPU-based, real-time pipeline which will be described in detail in the following chapter. We also describe a typical GPU implementation for coherent dedispersion systems.

CHAPTER 3

GPU-BASED TRANSIENT DETECTION PIPELINE

Fast transient surveys generate a large volume of data, which is not cost-effective to store, and therefore have to be conducted in real-time, where data from a telescope, which is continuously observing the sky, is streamed to a processing backend. In section 1.5 we have described a generic fast transient search process, and in this chapter we introduce our GPU-based transient detection pipeline. We start off by examining the current state-of-the-art of similar pipelines, and then discuss in detail the various processing stages of our pipeline, including RFI mitigation, dedispersion, event detection and classification, as well as data quantisation and persistence. These stages are encapsulated as a standalone framework which can be used in offline mode, for processing archival data, as well as within an online application with additional real-time capabilities. It uses the incoherent dedispersion kernel described in the previous chapter. We conclude by analysing the clustering and classification stage of the pipelines, and produce performance benchmarks.

3.1 Transient Detection Pipelines

Conventional pulsar surveying machines were essentially baseband recorders, where raw voltage data were recorded to disk and then processed offline on conventional compute clusters (for example, see [Voûte et al., 2002, Stars et al., 2002]). The need for real-time surveys has resulted in a conceptual shift, where most of the processing is now performed online and only data containing interesting events is recorded to disk. The new generation of surveying pipelines require a considerable amount of processing power, motivating the adoption and investigation of alter-

native compute platforms. Standard commercial, off-the-shelf (COTS) compute clusters are still popular. [Roy et al., 2009] have developed a fully real-time 32 antenna, 32 MHz dual-polarisation backend for the GMRT, which acts as a real-time correlator and beamformer, as well as a baseband recorder. Extensions for online transient detection are then discussed in [Bhat et al., 2013].

The Advanced Radio Transient Event Monitor and Identification System (ARTEMIS¹) [Serylak et al., 2012] was the first pipeline to explore the use of GPUs for transient detection systems, using the dedispersion kernel developed by [Magro et al., 2011] and eventually optimised further by [Armour et al., 2012]. This is implemented as a PELICAN pipeline. The Pipeline for Extensible Lightweight Imaging and CALibrationN (PELICAN²) framework is an efficient, lightweight, C++ library for processing data in quasi-realtime that separates data acquisition from data processing and output, allowing the scalability and flexibility to fit in different scenarios. ARTEMIS was mainly developed for deployment on LOFAR international stations, where the coarsely channelised, beamformed data are streamed toward a number of servers, each processing a subset of the beams. The processing stages include: data reception and interpretation, RFI filtering, polyphase-filter bank channelisation, dedispersion and event detection. The dedispersion stage is the only one which is performed on GPUs. A similar system is being developed for the Parkes radio telescope [Barsdell et al., 2011].

The use of FPGAs for online transient detection is also being investigated by [D’Addario et al., 2013], who are developing an FPGA-based, real-time backend for ASKAP, to act as the CRAFT processing platform. They use a Pico Computing EX-500³ backplane and up to 6 M-501⁴ FPGA modules, 4 of which dedisperse and detect 9 of 36 304 MHz dual-polarisation beams, each channelised into 1 MHz frequency channels, over a range of 442 DM trials, with 0.9 ms time resolution. We compare this system with the one presented in this chapter in section 3.9

3.2 Pipeline Overview

We consider the case where beamformed data are processed to extract astrophysical radio bursts of short duration and designed and implemented a generic, standalone, scalable, high throughput, GPU-based transient detection pipeline which can be

¹ <http://www.oerc.ox.ac.uk/research/artemis>

² <https://github.com/pelican/pelican>

³ <http://picocomputing.com/ex-series/ex-500>

⁴ <http://picocomputing.com/m-series/m-501>

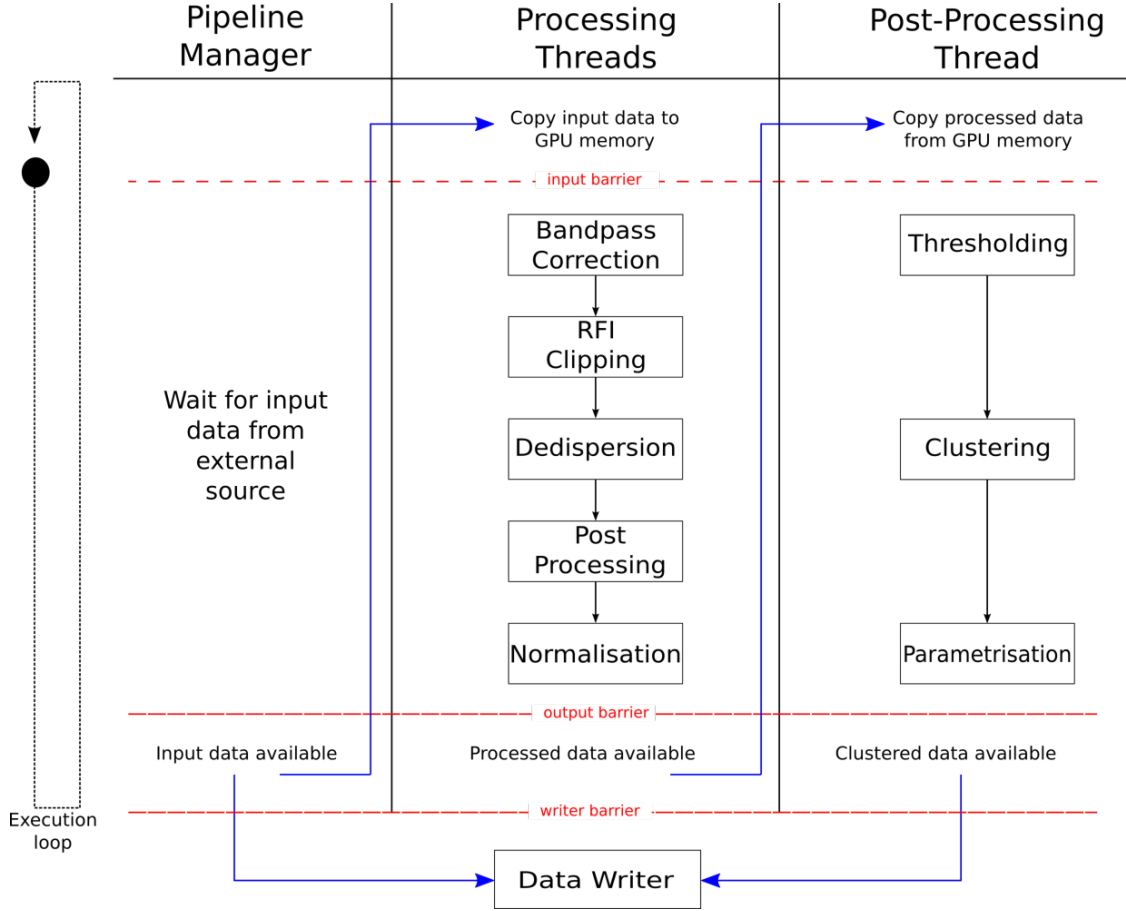


Figure 3.1: An extension of the architecture presented in figure 2.3. A processing thread is initialised for every input telescope beam which needs to be processed. These are processed independently and in parallel. The data writer can be triggered either from the pipeline manager in situations where the entire data series needs to be written to disk, or the post-processing thread when an interesting event is detected in the processed data. This will quantise, encode and write to disk the entire unprocessed input buffer, including data from all the beams being processed. Arrows in blue show data buffer movement (copies to/from GPU memory or across host memory buffers) while the dashed red lines are CPU barriers required to synchronise the three processing stages.

deployed on telescope backend processing servers for real-time transient surveys. We extended the pipeline design adopted in [Magro et al., 2011], described in detail in chapter 2, to include a fast buffering system, the possibility of writing streamed data to disk at different stages in the pipeline, the ability to process multiple beams concurrently and an online candidate selection mechanism to separate RFI events from astrophysical ones. The main design emphasis was high-performance and scalability across many beams.

The high-level architecture of this pipeline is depicted in figure 3.1. Data

acquisition is performed by a custom external data interpreter which forwards a data buffer containing either filterbank data or a voltage series which has been channelised into finer frequency channels. This buffer can contain data from multiple telescope beams which are distributed to multiple processing thread instances, one per thread. Each thread is associated with a single GPU device, however each GPU can have multiple processing threads assigned to it. These threads execute independently from each other. Once data becomes available each thread is responsible for copying the input buffer segment associated with it to GPU memory. This scheme maximises PCIe bandwidth on multi-GPU systems. Once the copies finish the processing threads pass through several processing stages: bandpass correction, RFI clipping, dedispersion and optional post-processing and normalisation. The CPU imprint of these threads is almost negligible and mostly consists of CUDA kernel timing and synchronisation, as well as a one off p^{th} order polynomial fit over N_c values per iteration, where N_c is the number of frequency channels. The dedispersed time series are then copied back to GPU memory and passed to the post-processing thread where they are thresholded, clustered and classified. Any data points belonging to interesting clusters are written to disk, together with the unprocessed data buffer after being quantised to 8 or 4 bits. There is also the possibility of writing the entire data stream to disk, including buffers without interesting events, after passing through an encoding and quantisation stage, provided that the disk drives can manage the data rate. All operating parameters are provided by an XML configuration file.

This architecture has several in-built assumptions and limitations, all of which help to speed up certain aspects of the workflow. The pipeline works on channelised data which are stored in beam/channel/time order in GPU memory, with time changing the fastest, and thus assumes that the data have been channelised and transposed during an earlier stage. For real-time systems the transpose is performed in chunks by the external data interpreter, as described in section 4.2.1, whilst channelisation is assumed to have been performed on the digital backend prior to packetisation and streaming. A GPU implementation of a polyphase filter bank channeliser to generate finer frequency channels would be a required addition to the pipeline in cases where the backend lacks this functionality, or only coarse-grained frequency channels can be streamed, as this would severely limit the sensitivity of transient searches due to signal smearing (for example, see [van der Veldt, 2011]).

Beams are assumed to be identical (same observation and survey parame-

ters), however this can easily be extended to allow the possibility of performing different operations on each beam. Also, having the capability to write data to disk from either the pipeline manager or post-processing thread requires a considerable amount of host memory, since an input buffer needs to remain accessible for three iterations. This feature is implemented as a triple buffering scheme, with buffer pointers updated after each iteration.

All GPU buffers are allocated once during the initialisation phase (per processing thread). The input and output buffers are recycled across different kernel launches to maximise use of available GPU memory, which is a limiting resource. Also, even though we could get a potential speedup and possibly a reduction in memory requirements, by combining some of the GPU kernels into monolithic functions, separating functionality this way makes it easier to add intermediary stages as well as disabling certain functionality when not required.

3.3 RFI Mitigation

Terrestrial RFI, and methods to try and mitigate or correct for it, is one of the major problems in transient surveys, especially for radio telescopes which are close to urban centres. Several methods have been investigated to cater for this (see [Hodgen et al., 2012] for a comparison of RFI mitigation techniques for dispersed pulse detection), the major challenge being to try and remove as much RFI as possible without affecting any true astrophysical dispersed pulses which might be present in the data. For our pipeline we implement a simple RFI-thresholding process which limits the amount of strong, bursty RFI, whose effectiveness can be tweaked by applying different thresholding factors. Any undetected RFI will result in incorrect detections after dedispersion, and although these are never welcome, it would be more advantageous to allow some low-power RFI to seep through rather than increasing the probability of clipping astrophysical events. The detection and classification stage should then be able to discern between real and terrestrial signals. Not performing any RFI mitigation would result in a large number of incorrect detections as well as a higher variance in the data, which lowers the probability of detecting weak astrophysical signals. The thresholding process consists of three stages as described below.

3.3.1 Bandpass Correction

A bandpass is a telescope’s response function across the frequency band being observed. For an ideal telescope this would be flat, however several effects, including the behaviour of the channelisation algorithm used, can result in different bandpass shapes. To properly mitigate narrowband RFI the bandpass shape needs to be modelled in order to detect isolated high power frequency channels. This process can either be performed offline prior to the observation, after which the model parameters are passed to the pipeline, or online where the bandpass is computed once per data buffer being processed. The former scheme is ideal for telescopes which have a well behaved and time invariant response function (although the power levels will change during the observation with variations in sky and telescope noise) whilst the latter is beneficial for experimental setups where the bandpass can change during the observation. Another approach is to model the bandpass during the first iteration in an online observation and then add weighted incremental updates to the model during the observation. For our implementation we decided to perform a bandpass fit at every iteration due to the dynamic nature of the setup at the BEST-II array (as described in section 4.3).

Once a data buffer is available in GPU memory a p^{th} -order polynomial is fitted to an averaged bandpass, providing a smoothed, approximate description of the telescope’s response to the frequency band being processed. Accumulation and averaging is performed on the GPU, mainly consisting of a reduction sum across the frequency channels, resulting in N_c values which are copied back to CPU memory. The resulting bandpass is then fitted using least-squares. The root mean square error (RMSE) between the fitted bandpass and the averaged, interpolated bandpass is then computed and used as a thresholding factor for the channel thresholding stage, whilst the mean and standard deviation are used for the spectrum thresholding stage. The fitted bandpass is then subtracted from all the spectra on the GPU, generating corrected spectra, having the effect of equalising the telescope’s response, as well as moving the mean to, or near, 0.

Online, iteration-based fitting can be ineffective if significantly powerful narrowband RFI is present, as this will distort the fit and generate incorrect thresholds. Such channels are usually intermittently or permanently on, depending on the source of the noise. For this reason these channels need to be masked prior to fitting and their values interpolated from neighbouring channels when calculating the averaged bandpass. Also, a smooth telescope response function can generate a very smooth averaged bandpass when accumulated over a large time frame (order

of seconds, depending on the sampling rate) resulting in a very low RMSE value after fitting. In these situations using the mean and standard deviation of the fit can generate better thresholds.

3.3.2 Channel Thresholding

Each frequency channel should have a uniform power level across short time spans. Sudden changes can generally be attributed to narrowband RFI (or alternatively, a very strong astrophysical burst). Strong, short duration, narrowband signals can result in incorrect detections across a wide DM range after dedispersion. If these signals are frequent and occur throughout the frequency range they could lead to detections exhibiting similar properties to dispersed pulses. For this reason such signals need to be removed from the data prior to dedispersion. If the RFI environment is properly modelled then persistently noisy channels can be masked before an observation. Also, if the duration of short duration signals is known a priori then a windowed thresholder can be implemented with window length, W_c , matching the signal's duration. When applied to the data buffer any such window having a mean value greater than the channel's threshold can be removed. Such a scheme will also remove longer duration signals in segments of length W_c . A small W_c can be used to implement a more generalised thresholder based on the same principle, where long duration signals will be removed one window length at a time. An optimal value for W_c , as well as the associated threshold, will depend on the RFI environment on-site, the width of astrophysical transients being searched for and the observing parameters, and can be empirically set through test observations.

We implemented this functionality using two successive GPU kernels. The first kernel applies a non-overlapping sliding window to each frequency channel, thereby partitioning them into chunks of width W_c . The mean of each chunk is calculated and compared to a pre-calculated channel threshold. If this threshold is exceeded then it is flagged for removal. Neighbouring blocks are also flagged so as to include the edges of long signals which might not be aligned with the partitioned grid. The list of flags is then used in the second kernel which replaces the values of flagged chunks with the frequency channel's fitted bandpass value. This is preferred to setting these values to 0 since the statistical properties of the data are maintained. For every frequency channel, this implementation takes the form of

$$\forall_c \in C_v : M(c) < B_\sigma \Rightarrow (\forall_i \in c : c_i = B_v) \quad (3.1)$$

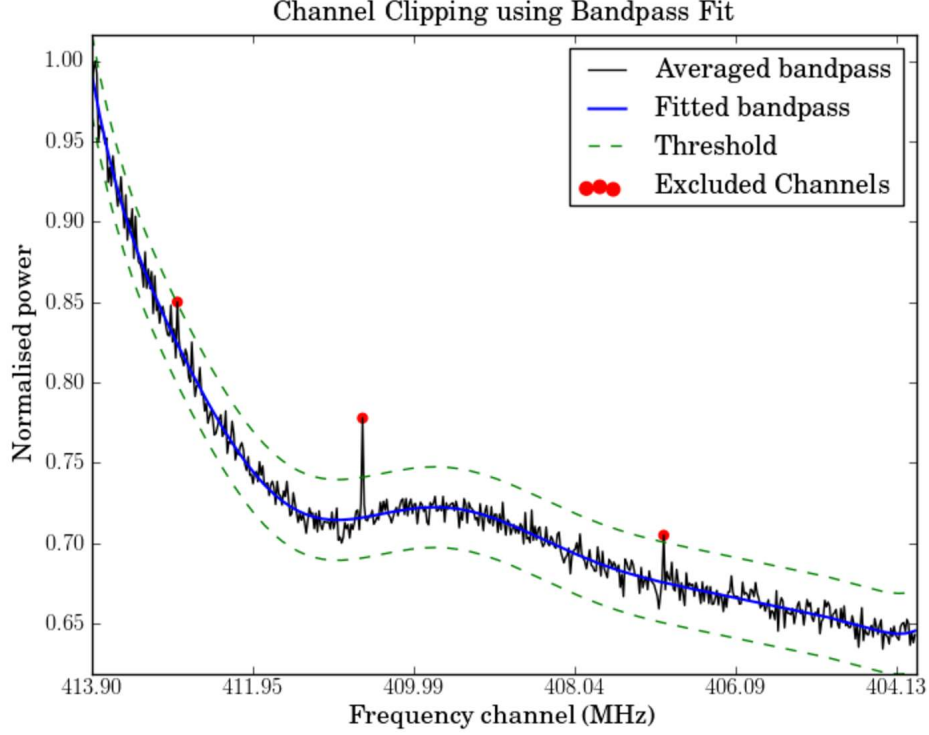


Figure 3.2: The averaged bandpass (black) is fitted with a p^{th} order polynomial (blue) and the RMSE of the two is used to generate channel clipping thresholds (green). This threshold is then applied to subsets of spectra and any chunk exceeding it will be replaced with the channel’s fitted bandpass value, as is the case for the channels marked in red in this plot.

where C_v is the set of partitions of length W_c for frequency channel v , $M(c)$ is a function which returns the mean of the partition, B_σ is the threshold value and B_c is the fitted bandpass value for channel v .

Figure 3.2 shows a one iteration snapshot during bandpass fitting, where the averaged bandpass is fitted with a p^{th} order polynomial and the RMSE of the two is used to generate the channel clipping thresholds in green. This threshold is then applied to subsets of spectra of size W_c and any chunk exceeding it will be replaced with the channel’s fitted bandpass value, as is the case for the channels marked in red in this figure. The lower threshold is required for when input data are lost, such as packet loss in real-time mode.

3.3.3 Spectrum Thresholding

Broadband RFI generally has the least impact on the dedispersion process as most of the detections will be concentrated at a DM of 0 pc cm^{-3} , which can however span over a wide range for long duration pulses. The general mitigation mechanism for broadband pulses is to ignore all detections for DM values less than a lower limit D_{\min} . We decided to implement a simple spectrum thresholding mechanism which removes strong broadband pulses to reduce the number of detections that need to be clustered in the post-processing stage. This process is performed on the GPU as well. The mean of each full-band spectrum is calculated and compared to a pre-calculated spectrum threshold. If the mean exceeds this threshold then the entire spectrum is replaced with:

$$\forall_c \in C : S_c = S_c - (S_c - B_c) \quad (3.2)$$

where C is the set of all channels, S is the entire spectrum and B is the fitted bandpass. This has the effect of removing the noise induced by RFI whilst maintaining the background properties of the affected signal. Dispersed pulses should not be affected by this thresholding, unless the threshold is exceptionally low, since their power is distributed across multiple spectra shifted depending on the pulses' DM value. The threshold is empirically set such that time spectra are rarely affected by this stage, although this will depend on the RFI environment.

3.3.4 RFI Mitigation Test

To demonstrate the effectiveness of the implemented RFI mitigation algorithms, a simulated filterbank dataset containing a dispersed pulse with a DM of 120 pc cm^{-3} , as well as narrow and broadband RFI, was generated. The simulated observation is centred at 413 MHz having a bandwidth of 10 MHz channelised into 512 frequency channels. 1.6 s of random noise with $\mu = 0$ and $\sigma = 1$ were first generated to which a 10 ms square, dispersed pulse having a S/N of 10 was added. Narrowband RFI (78 kHz, periodic signal with a period of 6.5 ms) and a broadband 2 ms pulse with an S/N of 10 were also added. Finally the data were multiplied by a bandpass shape modelled on the BEST-II beamformer output (discussed in the next chapter). Figure 3.3 shows the resulting waterfall plot, together with the accumulated bandpass and summed bandwidth. The simulated data were then passed through the RFI mitigation stages. The dispersed pulse didn't trigger any thresholding mechanism, whilst the narrowband RFI was replaced with the value

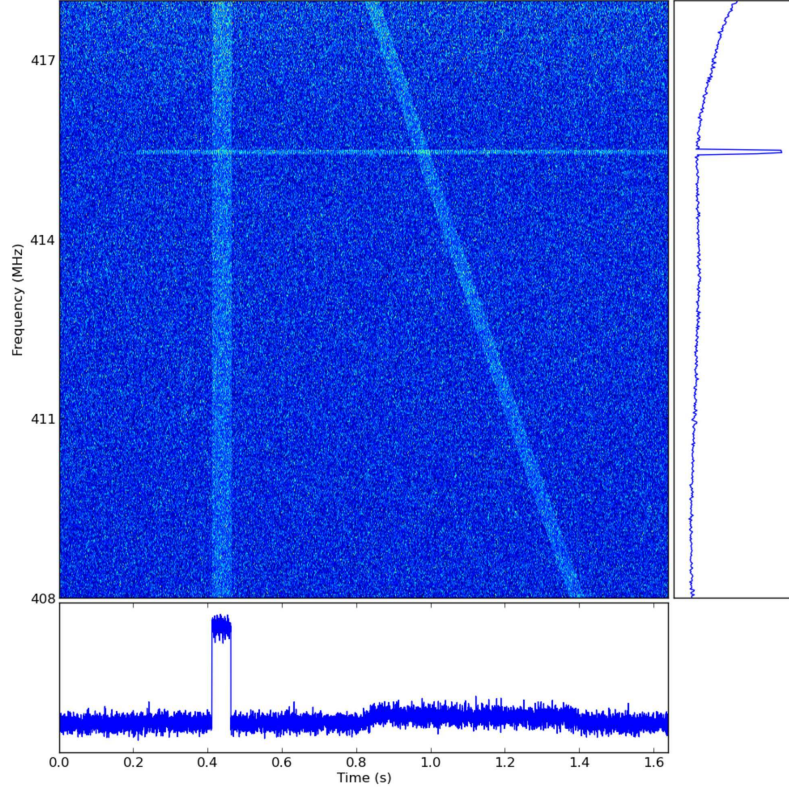


Figure 3.3: Simulated data file containing narrowband, broadband and dispersed signals, used to test the RFI mitigation stage. All these signals are clearly visible in the central waterfall plot, while the accumulated bandpass is shown on the right and summed bandwidth in the bottom plot.

of the fitted bandpass at the noisy frequencies. The broadband pulse was replaced by the value of the noisy signal, its difference from the bandpass values subtracted. A channel block length of 512 was used, while the channel threshold was set to 7 and spectrum threshold to 5. The resulting output is shown in figure 3.4

3.4 Dedispersion

Dedispersion is by far the most time consuming process in the entire pipeline. Its implementation underwent a series of optimisation runs with evolving GPU architecture, as discussed in chapter 2, where implementation details, as well as performance benchmarks, were presented. When integrating this kernel with a real-time pipeline several factors need to be taken into consideration, including how to maximize the use of available GPU memory and how to handle inter-buffer overlaps due to DM shifts.

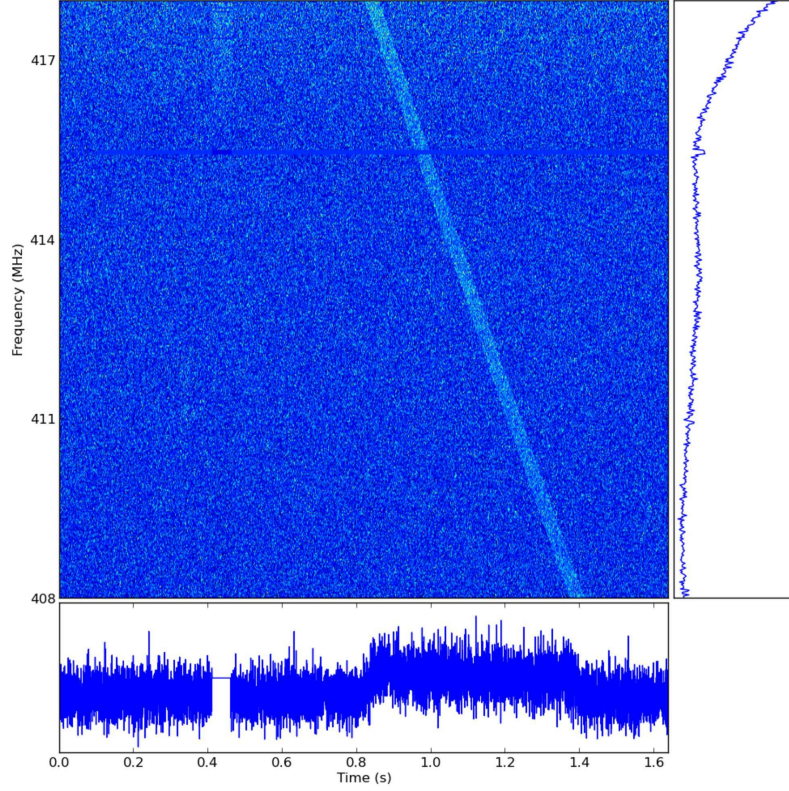


Figure 3.4: The simulated data from figure 3.3 was passed through the RFI mitigation stage and the resultant output is shown in this plot. Both the broadband and narrowband signals were mitigated by the implemented algorithms, whilst the dispersed pulse was left untouched.

Maximising the number of spectra which can be processed in a single iteration increases the compute to copy time ratio since high PCIe transfer rates can only be achieved with large transfer sizes. The dedispersion kernel is the only one which requires a separate output buffer to store the dedispersed time series, all the other stages process the buffer in place. A simple calculation can be used to compute this value:

$$N_s = \frac{M \cdot b/32 - (\Delta t_{\max} \cdot N_c) - \alpha}{N_{\text{DM}} + N_c} \quad (3.3)$$

where N_s is the number of time samples, N_c is the number of frequency channels, N_{DM} is the number of DM trials, M is the amount of GPU global memory in 32-bit words, Δt_{\max} is the dispersion shift in time samples between the edges of the observing band for the largest DM value, which will be referred to as maxshift throughout this chapter, b is the input bitwidth and α represents other smaller buffers which are required (such as the shift buffer for dedispersion and fitted bandpass buffer for bandpass correction).

Maxshift represents the extra number of spectra required to process the data in one iteration. These extra spectra still need to be dedispersed, so during the following iteration they are first copied to the beginning of the buffer and the next input buffer is appended to it. Due to the fact that the data have been transposed, this actually needs to be performed for each frequency channel separately, so a series of asynchronous memory transfer calls are issued. Also, in order to avoid extra computation in the dedispersion kernel the shifts in unit sample intervals per DM and frequency channel are computed once during thread initialisation and transferred to GPU memory. These shifts are stored in channel/DM order so as to assure coalesced memory access.

3.5 Signal Post-processing

The dedispersed time series are then passed through a post-processing stage, which smooths and normalises the data. This is mostly done using two techniques:

Median Filtering: Noisy, isolated outliers in the time series can be suppressed by replacing the value y_m with one which is calculated using neighbouring points:

$$y_m = operation\{x_i, i \in w\} \quad (3.4)$$

where w represents a neighbourhood centred around the location m , and *operation* is the averaging function used. We chose to use the median of the neighbourhood points, since it better preserves the edges of the signal than the mean. We have implemented a windowed-median filter on the GPU, where thread blocks are split across a 2D grid with each row processing one dedispersed time series, for a single DM value, partitioned along the columns of the grid. Each thread block loads a subset of the series to shared memory, including some overlapping values at the edges, and then each thread computes the median of its associated data point neighbourhood and stores this value back to global memory. Finding the median of a sequence of values requires a partial sort (of length w) so it can be inefficient since thread warps will generally have to divert branch execution during comparisons. A w value of 5 or 7 generally gives a good ratio between performance and filtering efficiency, however this also depends on the time resolution. It should be noted that median filtering will limit the detectability of short duration and unresolved signals, with widths close to the sampling time.

Detrending and Normalisation: The mean power of the incoming data stream can change gradually in time, the rate of which depends on the cause of this change, such as steady temperature changes in telescope electronics, sky temperature variations, or an astrophysical radio source moving toward, or away from, the beam centre. This effect can be alleviated by subtracting a best-fit line to the dedispersed time series, which effectively centres the series to a mean of 0. The detrending process is also performed on the GPU where thread blocks are assigned a single dedispersed time series to process, which essentially compute a best-fit line using linear regression. The kernel requires two passes of the data, one to calculate the regression parameters and another to subtract the trend-line from the series. This is performed by having the thread block move in a non-overlapping window fashion across the series, with each thread accumulating values locally. The final value is then computed collaboratively using a reduction sum mechanism. During the second pass the standard deviation is also computed, so by adding a third pass the data can be normalised and prepared for thresholding.

After this stage, the post-processed dedispersed time series are copied back to CPU memory and forwarded to the event detection stage, which is started as soon as all the beams have been processed (and a new input data buffer is available for processing).

3.6 Detection and Candidate Selection

During the first stage of event detection, the dedispersed time series are thresholded using a suitable threshold value ($n\sigma$). This value should be low enough to allow low S/N pulses to pass through, even at the cost of incorrect RFI detections, which will be filtered in the clustering stage. A list of detections is generated for each beam, containing (time, DM, intensity) triplets. Astrophysical transients, as well as RFI signals, will result in a number of entries in this list which should be grouped together and treated as a single candidate. This is the main function of the clustering stage.

We start by applying a density-based clustering technique, DBSCAN [Ester et al., 1996], to group neighbouring data points together. Its definition of a cluster is based on the underlying estimated density distribution of the dataset. The shape of the clusters is determined by the choice of the distance function for two points. The *Eps-neighbourhood* of a point p , denoted by $N_{Eps}(p)$ is defined

by $N_{\text{Eps}}(p) = \{q \in D \mid \text{dist}(p, q) \leq \text{Eps}\}$ where Eps is an argument defining the neighbourhood extent of a point, D is the set of points and $\text{dist}(p, q)$ is the distance function for points p and q . DBSCAN distinguishes between two types of cluster points, those inside the cluster (*core points*) and other at the border of the cluster (*border points*) which generally have less points in their neighbourhood. A point p is *directly density-reachable* from q if $p \in N_{\text{Eps}}(q)$ and $|N_{\text{Eps}}(q)| \geq \text{MinPts}$, where MinPts is the minimum number of data points. A point p is *density reachable* from a point q if there is a chain of points p_1, \dots, p_n , $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i . Border points within the same cluster might not be density-reachable from each other, but they are *density-connected* if there is a point o such that both of them are density-reachable from o . Following these definitions, a *cluster* is defined as a non-empty subset of D satisfying the following two conditions:

- *Maximality*: $\forall p, q : p \in C \wedge (q \text{ is density-reachable from } p) \Rightarrow q \in C$
- *Connectivity*: $\forall p, q \in C : p \text{ is density-connected to } q$

Any points which do not belong to a cluster are regarded as noise: $N = \{p \in D \mid \forall i : p \notin C_i\}$, where N is the set of noise points, C_1, \dots, C_k are the clusters in D and $i = 1, \dots, k$ with k being the total number of clusters. This technique has several advantages which makes it a suitable candidate for clustering dedispersed thresholded detections:

1. it does not require a seed to specify the number of clusters in the dataset, which is a desirable property since the number of events occurring within a time-frame, be they astrophysical or RFI, is unknown unless a cluster approximation step is performed beforehand
2. it has a notion of noise
3. it is also capable of separating overlapping clusters having different density distributions, such as when an RFI signal overlaps a transient event, if the input arguments are sensitive enough

The main drawback of this technique is its runtime complexity, dominated by the neighbourhood calculation for every point, which is of order $\mathcal{O}(N^2)$ unless an indexing structure is used or a distance matrix is computed beforehand, however this needs $\mathcal{O}(N^2)$ memory, which is unfeasible for large datasets. To counter this

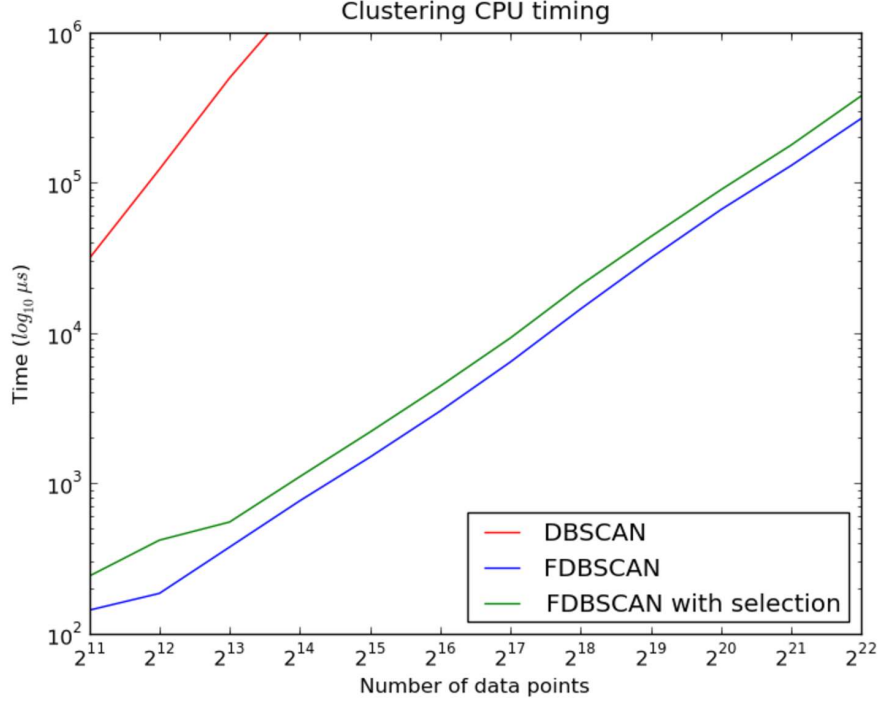


Figure 3.5: The significant difference in runtime between DBSCAN and FDBSCAN, with candidate selection only increasing this by a small factor.

we implement a faster, albeit less accurate, version of the algorithm, FDBSCAN [Zhou et al., 2000], which only uses a small subset of representative points in a core point’s neighbourhood as seeds for cluster expansion, reducing the number of region query calls. The representative points are chosen to be at the border of a core point’s neighbourhood, two for each dimension, one for each direction when placing the core point at the origin. Due to this approximation, some points might be lost, and in some cases clusters might be split apart, however the probability of this happening is very low (see [Zhou et al., 2000]).

The distance function used in our implementation assigns a different value to each dimension, and thus requires three different values for Eps : (T_ϵ , S/N_ϵ , DM_ϵ). Detections after dedispersion will have a specific shape along the time dimension due to incorrect dedispersion, with higher DM values detecting events before lower ones. The width of a cluster will also reflect the true pulse width, being narrower near the true DM. The value of T_ϵ should depend on the lowest pulse width being searched for. A higher T_ϵ might result in pulses close in time to be fused together into one cluster. The value of S/N_ϵ and DM_ϵ should be large enough to allow clusters to encompass the entire range, whilst allowing DBSCAN to distinguish between core and border points.

The output of the clustering stage is a list of detected clusters. The main challenge is then to discern between RFI-induced clusters and transient candidates. In the case of transients the highest S/N detections will be centred around the pulse's true DM, diminishing in power when moving away from this value until the threshold level is reached due to dedispersion at an incorrect DM value. Following [Cordes and McLaughlin, 2003], assuming a rectangular bandpass function, which should resemble the telescope's bandpass shape after bandpass correction, a Gaussian-shaped pulse with a width at FWHM of W in milliseconds, the ratio of the measured peak flux density $S(\delta\text{DM})$ to true peak flux S for a DM error δDM is

$$\frac{S(\delta\text{DM})}{S} = \frac{\sqrt{\pi}}{2} \zeta^{-1} \text{erf } \zeta \quad (3.5)$$

where

$$\zeta = 6.91 \times 10^{-3} \delta\text{DM} \frac{v_{\text{MHz}}}{W_{\text{ms}} v_{\text{GHz}}^3} \quad (3.6)$$

Comparing this model with a cluster's DM-S/N signature provides us with a classification mechanism. Our implementation performs the following steps for each detected cluster:

1. Generate its DM-S/N signature by collapsing the time dimension.
2. Smooth this signature by running a k -element moving average.
3. Find the DM value containing the highest number of detections and associated maximum S/N. If this DM value is less than 1.0 then it is assumed that it was caused by broadband RFI and the cluster is discarded.
4. Approximate the pulse's FWHM by computing the difference between the pulse's start and end time at the maximum S/N value.
5. Normalise DM-S/N signature.
6. Compute the analytical curve for incorrect dedispersion using equation 3.5.
7. Calculate the RMSE between the modelled curve and pulse's DM-S/N signature.
8. If the RMSE exceeds a preset threshold, then the cluster is discarded (RFI), otherwise it is classified as a potential candidate. This threshold can be set empirically through test observations or by using simulated data.

This procedure is most effective for detecting relatively strong pulses and differentiating them from RFI. The classification of clusters having a small number of detections can be incorrect if the width of the pulse cannot be determined. To increase the likelihood of detecting lower-S/N pulses, the detection threshold during the first stage of event detection should be lower than typically used for similar searches in order to allow more pulse detections to be clustered together. This will result in a higher number of background noise detections, however these will be filtered out by DBSCAN.

3.7 Data Persister

The pipeline can either write the entire incoming data stream to disk or dump data buffers containing interesting detections when triggered from the event detection stage, for future off-line processing. If the incoming data rate being processed is higher than the attached disk drives' write speed then the data have to be quantised first. The quantisation mechanism was developed for the pipeline's deployment at the Medicina BEST-II array (discussed in chapter 4) so the accepted data formats are currently limited to two types: signed 16-bit complex channelised voltage series which are quantised to signed 4-bit complex values, with two complex components packed into one byte, and the equivalent detected unsigned 32-bit single-precision floating point power series which is quantised to unsigned 8-bit bytes ([Backer et al., 1997] have thoroughly discussed the effect of quantising a data stream down to 4-bits).

Incoming voltage series typically (depending on the quantisation mechanism on the backend) follow a normal distribution with mean 0. When this voltage series is detected the distribution changes to a half-normal one (if X follows an ordinary normal distribution $N(0, \sigma^2)$ then $Y = |X|$ follows a half normal distribution). In both cases most of the data values will be concentrated around the mean, around $\pm 3\sigma$ for normal and 3σ for half-normal distributions. Regular quantisation schemes, which represent a signal using L regularly spaced levels in increments of $\epsilon\sigma$, will assign a single bit to each level. For example, if an 8-level, $\epsilon = 1$ scheme is used, the region between $[3\sigma, 4\sigma]$ will have a dynamic range equal to the region $[\mu, \sigma]$ even though the latter represents many more data points. Therefore, using a linear quantiser would result in a loss in sensitivity within the region centred around the mean. Also, any strong signals present in the data, be they astrophysical or RFI-induced, will be represented in the bins at the edges, potentially

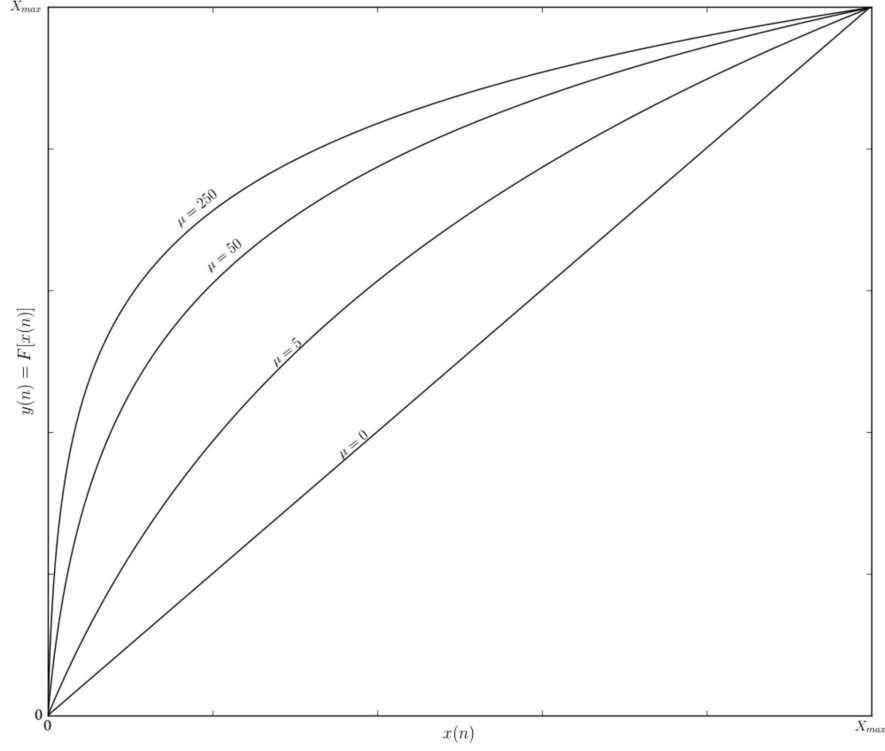


Figure 3.6: The mapping between input and output values for a given range when applying the μ -law algorithm

reducing the S/N.

An alternative method would be to use a logarithmic quantiser, where quantisation levels are distributed according to a logarithm function, resulting in finer resolution (smaller quantisation steps) around the signal mean. In our pipeline we implement a μ -law quantiser, adapted from the G.711.0 standard⁵:

$$y = X_{\max} \frac{\log \left[1 + \mu \frac{|x|}{X_{\max}} \right]}{\log [1 + \mu]} \text{sign} [x] \quad (3.7)$$

where x represents points in the data series, X_{\max} is the maximum value in this series, μ is the compression factor and y is the quantised data series. A higher compression factor results in more output bits being allocated to the high data point concentration range of the input distribution, as shown in figure 3.6, while figure 3.7 shows the effect the compression factor has when quantising signed 16-bit input values down to signed 4 bits. Logarithmic quantisers are more computationally intensive since for every input sample a logarithm needs to be calculated. We

⁵ See <http://www.itu.int/rec/T-REC-G.711>

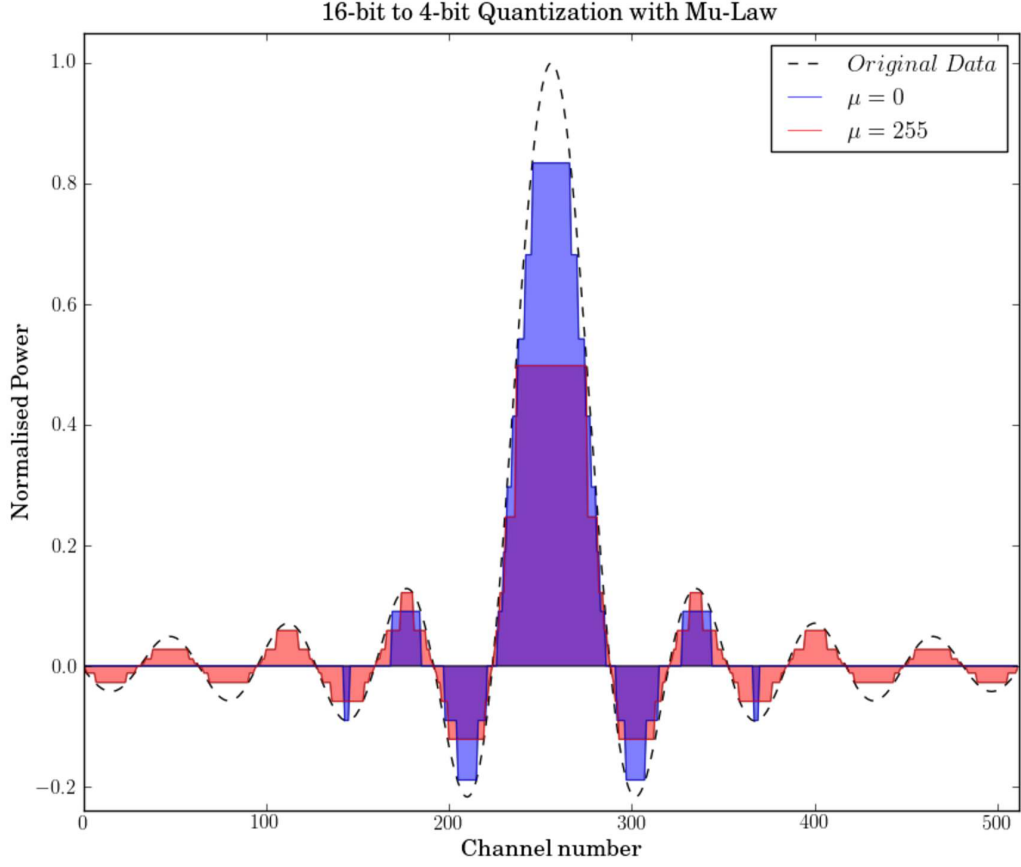


Figure 3.7: The effect of the compression factor has when quantising signed 16-bit input values down to signed 4 bits. When the compression factor is high most of the detail in the trailing curve is retained, at the expense of losing information at higher values. Applying this to channelised raw voltages would have the effect of allocating more bits to the region around the mean and clipping high-valued outliers.

counter this by pre-computing a log lookup table for the input data range, small enough such that it can be stored in cache.

The data buffer is first encoded using this mapping and then the output values are quantised and written to disk. This is performed on the CPU, so as not to interfere with the main processing pipeline and induce delays. The data series is split across multiple OpenMP threads, whose processing is interleaved with file I/O calls, in order to overlap CPU-processing and disk I/O. Figure 3.8 shows the linear performance scalability of our implementation when processing a single beam.

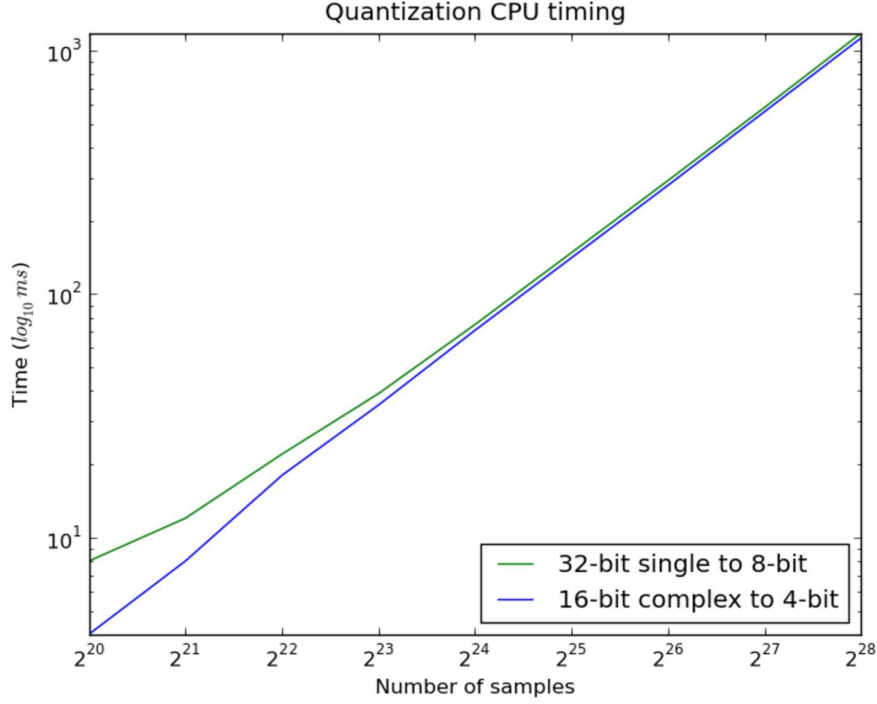


Figure 3.8: Scaling benchmarks of our CPU implementation when converting 32-bit single precision values to 8-bit and two 16-bit components to two 4-bit values packed into one byte.

3.8 Pipeline Analysis

We have described in detail the various processing stages composing our GPU-based transient detection pipeline. In this section we analyse its performance scalability and define upper bounds on the number of beams and DM trials which can be processed on a standard COTS GPU server. We limit this analysis to BEST-II observing parameters, as described in chapter 4, while in chapter 6 we analyse the scalability of this pipeline to a wider parameter range, with special emphasis to SKA₁ specifications. We also provide several figures of merit for a number of processing stages by evaluating them against simulated data over a wide parameter range.

We have developed a lightweight transient pipeline simulator, written in Matlab, which was used to generate test filterbank data containing both dispersed and RFI signals, as well as to act as a prototyping platform to test the applicability of different algorithms to all the stages within the pipeline. The generated pulses can be tracked as they pass through different processing stages and then the performance of the single pulse detection stages, as well as the RFI mitigation

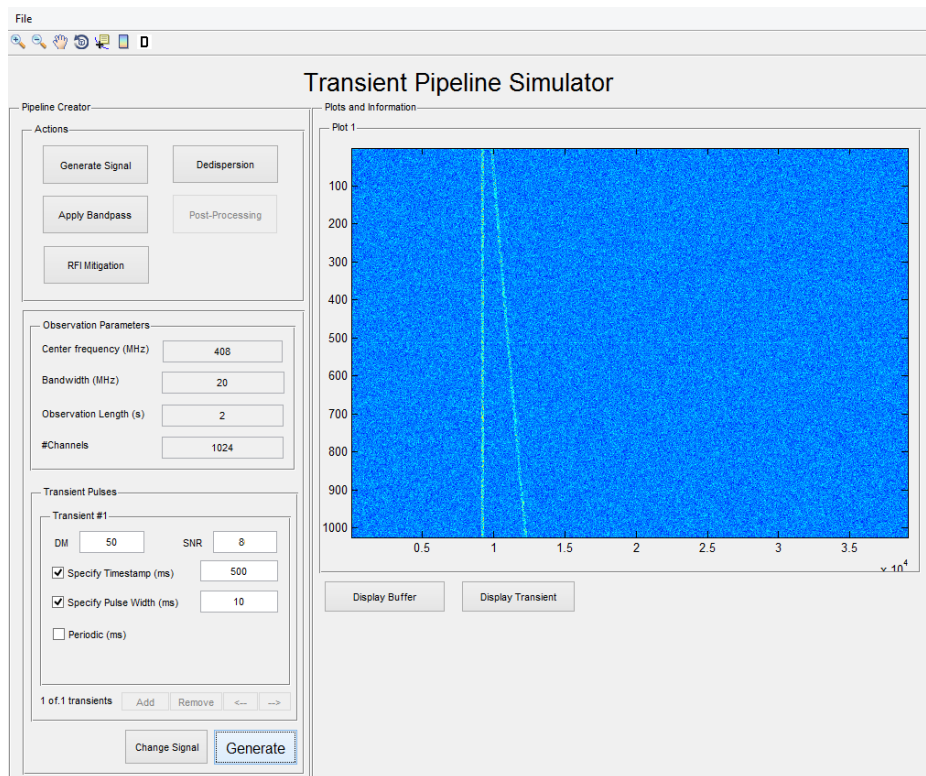


Figure 3.9: Screenshot of Matlab transient detection pipeline simulator which was used to analyse the algorithms implemented in the GPU pipeline and to generate simulated data for evaluation. The data generation stage is shown in this figure.

algorithms, can be measured using appropriate figures of merits. The aim of this tool is to provide a quick and easy way to test novel algorithms, and provide an intuitive Graphical User Interface (GUI) for quick visual inspection and plotting mechanism. Figure 3.9 provides a screenshot of the data generation stage. This tool was used to generate the test data described below.

3.8.1 Clustering and Classification Evaluation

The principal aim of a transient detection pipeline is to detect as many “interesting” signals as possible, over a wide parameter range, including DM, pulse width, S/N and pulse shape. The standard technique which is generally used is template matching, where boxcar functions of varying width are convolved with each dedispersed time series, the result of which is then thresholded. Each boxcar template has the effect of downfactoring the time series, thus increasing the S/N of wider

pulses. This process is compute intensive, with a time complexity of

$$\mathcal{O}(N_{\text{DM}}N_tN_s\log N_s) \quad (3.8)$$

where N_{DM} is the number of dispersion measure trials, N_s is the number of time samples and N_t is the number of templates being matched. This process is generally performed in the Fourier domain, meaning that a forward and backward Fourier Transform are required. The output of this process still needs to be clustered in some way such that detections belonging to the same event are grouped together. Resulting groups are then classified as either RFI or not, after which appropriate action is taken.

Matched filtering techniques were briefly discussed in section 1.5.1, where it was stated that narrower pulses are more easily detected than broader ones, however low-amplitude, broad pulses are more easily detectable than sharp, narrow pulses if its area is sufficiently large. There are, however, several downsides to this technique. First of all, as was already stated, it is a compute intensive process. Secondly, boxcar functions are generally used, and thus can be insensitive to pulses which cannot be approximated well by a top hat pulse, such as highly scattered pulses. Template matching also increases the number of detections when compared to standard thresholding, as multiple templates are generally used.

As we discussed in section 3.6, we apply a density based clustering technique on the thresholded dedispersed time series, which groups detections that were caused by the same event together. An unoptimised DBSCAN implementation, with an indexing structure for region queries, has a complexity of $\mathcal{O}(N_\sigma \log N_\sigma)$, where N_σ is the number of points with standard deviation greater than a preset threshold. FDBSCAN reduces the $\log N_\sigma$ factor to a scalar value defined by the number of representative seeds taken from a core point's neighbourhood which

Centre Frequency	418 MHz
Bandwidth	20 MHz
Frequency Channels	1024
Sampling Time	51.2 μs
DM	25 pc cm ⁻³
Dispersed pulse S/N	0.5 - 5, increments of 0.5
Dispersed pulse widths	0.1 - 102.5 ms (10 unique values)
Repetitions per event	10

Table 3.1: Simulated data parameters for event detection accuracy test

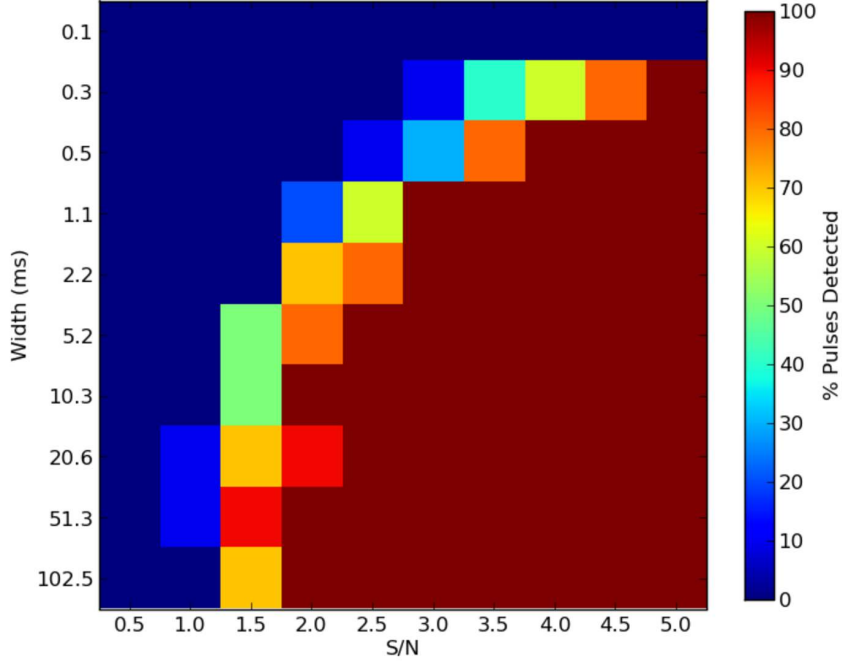


Figure 3.10: Output for clustering and classification accuracy test. 10 dispersed pulses for each combination were present in the dataset, the number of which were detected is show here. A pulse width of 0.1 ms is equivalent to ~ 2 time samples.

are used for cluster expansion. Therefore, our implementation has a complexity of $\mathcal{O}(N)$, which makes it considerably faster than template matching, and an ideal candidate for real-time detection. The cluster parametrisation and candidate selection stage have a negligible effect on execution time.

A simple metric which can be used to test the accuracy of the event detection stage is the percentage of pulses which are correctly detected and classified. To measure this, a simulated dataset was generated, the parameters of which are listed in table 3.1. A total of 100 different dispersed pulses, with a DM of 25 pc cm^{-3} , each repeated 10 times, were injected into a stream of Gaussian noise with mean 0 and standard deviation 1. The pulses are Gaussian shaped, with a value at the mean equal to $\sqrt{S/N}$ in each channel, with width at FWHM equal to the width of the pulse. This data set was then processed by the transient detection pipeline, dedispersed over 2048 DM trials with a DM step of 0.04 pc cm^{-3} . The RFI thresholding and post-processing stages were disabled. The event detection threshold was set to 4σ , resulting in a high number of background noise detections as well as better sensitivity to low S/N pulses. The detected pulses are shown in figure 3.10, where the number of pulses which were correctly detected and classified with respect to the original dataset are listed. Narrow pulses, of the order of 2

to 10 samples wide, were only fully detected for a pulse S/N of 4.0 or greater, while wider pulses were detected for pulse S/N as low as 2.0. This emphasises the fact that such a classification scheme is sensitive to pulses of varying widths and S/N. It should be noted that the S/N of wide pulses can be increased by integrating consecutive time samples together. Also, applying a median filter to the dedispersed time series will reduce the detectability of very narrow pulses, since they might be considered as noisy outliers.

An additional benefit of the classification stage is that it automatically detects the DM and width of the pulse, which are used to compute the analytical curve for incorrect dedispersion. Figure 3.11 shows the accuracy of this process with respect to accurate (a) DM detection and (b) RMSE. Wider pulses induce a wider DM error margin due to random fluctuations within the pulse peak. For this reason, the DM is approximated by calculating the median of all the values in the DM histogram having a normalised value greater than 0.75. As the pulse gets wider, random fluctuations also induce a higher error in the calculated RMSE value, resulting in a higher error margin.

3.8.2 RFI Mitigation

A simple metric which can be used to test the pipeline’s resilience to RFI events is the percentage of RFI events which pass through the event detection stage and are incorrectly classified as astrophysical events. In order to be able to come up with a value for a particular implementation, the actual number of RFI events which occur during an observation needs to be known. There are two ways in which this can be done. The first is to generate a simulated observation where all the RFI events, and other additional signals, are accounted for. The second method is to conduct a real observation of an “empty” area of sky with the telescope of interest, such that all events will be RFI-induced. For the latter method an accurate statistical model of the RFI environment is required, and the test observation should be conducted for enough time to obtain a statistically significant value. Here we use the former approach.

A simulated dataset was generated to test the pipeline’s resilience to RFI events, the parameters of which are listed in table 3.2. A total of 100 different broadband events, each repeated 10 times, were injected into a stream of Gaussian noise with mean 0 and standard deviation 1. A 97.6 kHz (5 frequency channels) constant narrowband signal was also included, as well as random narrowband events of 1 s duration, which should incur some detections since the GPU buffer

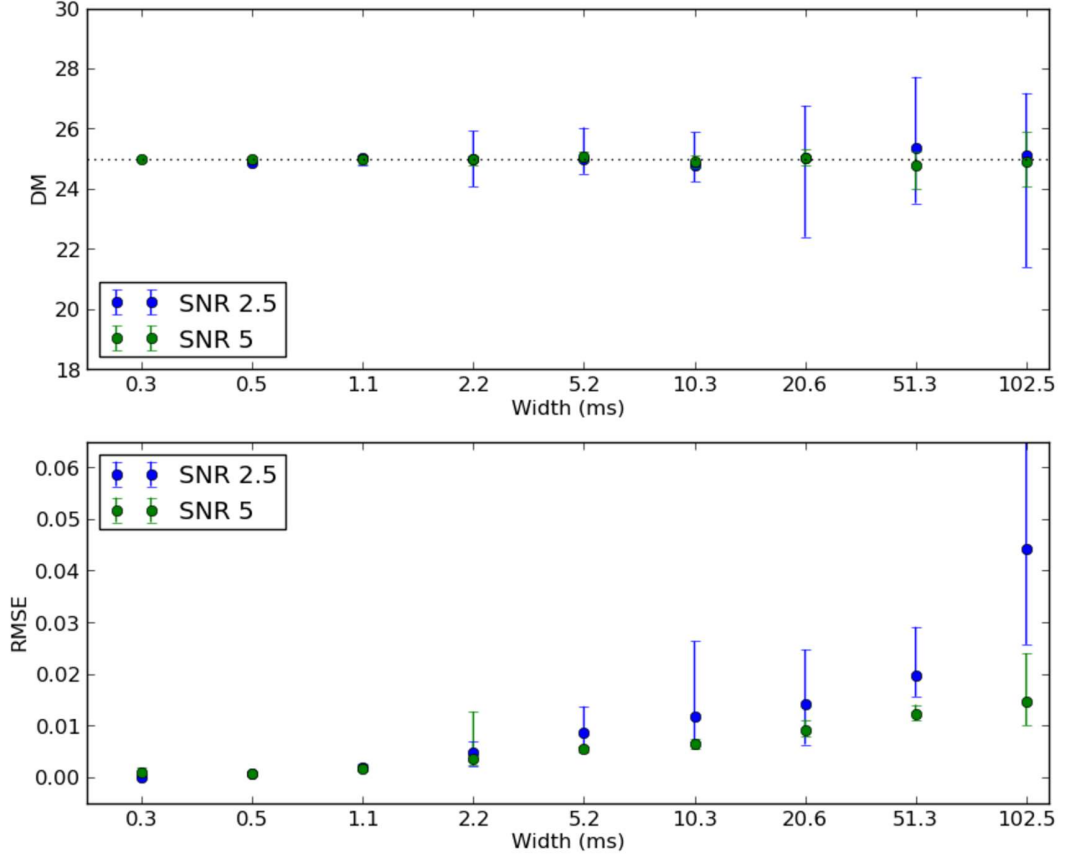


Figure 3.11: DM and RMSE values for the simulated pulses as calculated by the classification stage. Uncertainties increase as pulses get wider. Error bars represent the range between the maximum and minimum values.

used to run the test was 3.3 s long. A 6th-order polynomial bandpass was also applied to the frequency band, which decreases the response at the edges of the band. This data file was then processed by the transient detection pipeline, where the RFI mitigation, dedispersion and clustering stages were enabled. RFI thresh-

Centre Frequency	418 MHz
Bandwidth	20 MHz
Frequency Channels	1024
Sampling Time	51.2 μ s
Broadband RFI width range	1 - 30 ms (10 unique values)
Broadband RFI S/N	1 to 10, increments of 1
Repetitions per event	10
Narrowband RFI frequency	97.6 kHz

Table 3.2: Simulated data parameters for RFI thresholding accuracy test

olding parameters were set such that dispersed pulses would not be affected (see previous section). The time series data were dedispersed over 2048 DM trials, with a DM step of 0.04 pc cm^{-3} . All the detections were then clustered and evaluated.

Most of the simulated signals did pass through the RFI mitigation stage, as the thresholds were set a large value, however 100% of the broadband signals were correctly classified as RFI, whilst a single cluster caused by a narrowband burst was incorrectly classified as an astrophysical event. This emphasises the fact that a clustering technique, coupled with a classification mechanism, is a viable and accurate candidate for event detection. It should be noted that the simulated RFI events were simple signals, in that they are vertical or horizontal lines in time and frequency space. More complicated artificial signals can induce erroneous positive detections, and this points towards the need for a more robust classification mechanism. Pattern matching and machine learning techniques can be employed for this, however any supervised learning algorithm would require a labelled training set consisting of RFI and non-RFI signals. The RFI environment of a radio telescope can be determined by observing an empty area of sky, where all detections would be RFI-induced. All the resultant detections can then be used as a training set.

3.8.3 Performance Benchmarks

The transient detection pipeline can be thought of as a soft real-time system, where all the parallel stages should keep up with the incoming data stream for maximum quality of service, however if for some reason one of them does not meet a processing deadline the system does not become unstable, but rather the processing of input data will be delayed until the pipeline progresses by one iteration. This has no effect when running in offline mode, however when deployed as a real-time system input data might be lost. These processing hiccups might happen when, for example, a very high-power RFI spike induces a large number of detections resulting in a large number of data points to cluster. The GPU processing times are fixed regardless of the quality of the data, so these issues are only applicable to CPU threads. For this reason, a pipeline iteration should be limited by the GPU processing time. The scaling performance for the two most time consuming stages on the CPU, the clustering and quantisation stages, are shown in figures 3.5 and 3.8 respectively, each performed by a single thread. The number of samples which need to be quantised for a single pipeline iteration is fixed ($f_{\text{bw}} \times N_{\text{beams}} \times N_{\text{components}} \times t_s$), while the number of data points which need to be clustered depends on the number

of events present in the data stream.

The upper bound on the number of dispersion measure trials which can be processed in real-time depends on several factors, including: the observing bandwidth and centre frequency, the number of simultaneous beams which have to be processed, the DM step and the setup on which it is deployed. As a general rule, for a given setup, the number of DM trials is inversely proportional to the number of beams, which leads to a compromise between field of view and observable distance, assuming an appropriate DM step is used. We choose to evaluate the performance of this pipeline by using the observing parameters of the BEST-II array, discussed in chapter 4.

Table 3.3 lists the average timings for 10 pipeline iterations for all the processing stages for one pipeline iteration when processing eight 20 MHz beams, channelised into 1024 frequency channels and centred at 408 MHz. A simulated data file containing a 500 ms periodic pulse with 1% duty cycle and a S/N of 3 was input to the system, with the data copied to all the beams in 5 s buffers, this length being limited by the amount of GPU memory available. This results in about 18 clusters, with a total of 2^{16} points, per iteration. GPU timings are for a single GTX 660 Ti processing 4 beams while CPU timings are for all the beams when processed using a single output thread and quantisation thread. The host system on which the benchmark tests were conducted consists of 2 Intel Xeon E5-2630 2.3 GHz processors, 2 NVIDIA GTX 660 Ti GPUs with 3GB of GDDR5 RAM, 32 GB DDR3-1600 system RAM and a Fujitsu D3118 system board. The table is split into two columns which work in parallel, the GPU-based processing stages and the CPU-based processing stages, with data transfer to and from the two performed at synchronisation barriers. It is clear that the processing bottleneck is the dedispersion kernel, which takes up 93% of the GPUs processing time, while the rest of the kernels have a negligible effect on the overall running time of the pipeline. The number of DM trials which can be processed during an observation is limited by this as well, and currently this value is around 864 DMs.

3.9 Comparison with Other Work

In section 3.1 we discussed the current state-of-the-art for fast transient pipelines, with implementations ranging from conventional CPU-based systems to ones which use GPU or FPGA accelerators to speed up processing. We will not compare the performance between CPU- and GPU-based systems, as this has been thoroughly

Copy to GPU: 295.30 ms			
GPU		CPU	
Bandpass Fitting	36.79 ms	Thresholding	690.96 ms
RFI Filtering	74.97 ms	Clustering	1148 ms
Dedispersion	3863 ms	Classification	168.64 ms
Median Filtering	135.11 ms	Clusters to file	728.73 ms
Detrending	59.06 ms	Quantisation*	3532 ms
Copy from GPU: 155.97 ms			
Total iteration time: 4619.72 ms			

Table 3.3: GPU and CPU timings, averaged over 10 pipeline iterations, when processing 8 20MHz beams split between 2 NVIDIA GTX 660Ti cards. GPU timings are for a single GTX 660 Ti processing 4 beams while CPU timings are for all the beams when processed using a single output thread and quantisation thread.

* Quantisation is performed on a different CPU thread from the event detection stages

performed in other work (for example, [Barsdell et al., 2011]). No performance benchmarks have been published for the ARTEMIS system to date, so a direct comparison is not possible. It should be noted, however, that the principal ideology for these two systems is different. ARTEMIS uses the concept of “data blobs”, where small data buffers are passed through the CPU-based pipeline modules, after which the dedispersion module generates larger buffers to offload to GPU memory for processing. This increases CPU performance for certain algorithms due to higher cache coherency, however it also requires a number of additional memory copies amongst data blobs, and an additional buffering scheme. The CPU-based RFI clipper is also based on a median filter, which requires a partial sort and thus is one of the primary CPU bottlenecks. In our implementation, buffering is performed at the start of the pipeline and all processing stages are then executed on the GPU, resulting in less overhead, and is less coupled with the scheduling policy of the operating systems, which could result in a temporary degradation in performance.

[D’Addario et al., 2013] provide some performance benchmarks as well as a detailed architectural description, which is reproduced in figure 3.12. Dedispersion and detection is split across 4 Virtex 6 LX240T, whilst an additional FPGA performs an incoherent summation across antennas and also sums the two polarisations. Each dedispersion FPGA processes 9 power beams, split into 304 channels, with a time resolution of 0.9 ms, for up to 442 DM trials in real-time. Ignoring the operations required for summation and detection, this results in approximately 5.7×10^9 operations per second for dedispersion spread over 4 FPGAs, at peak

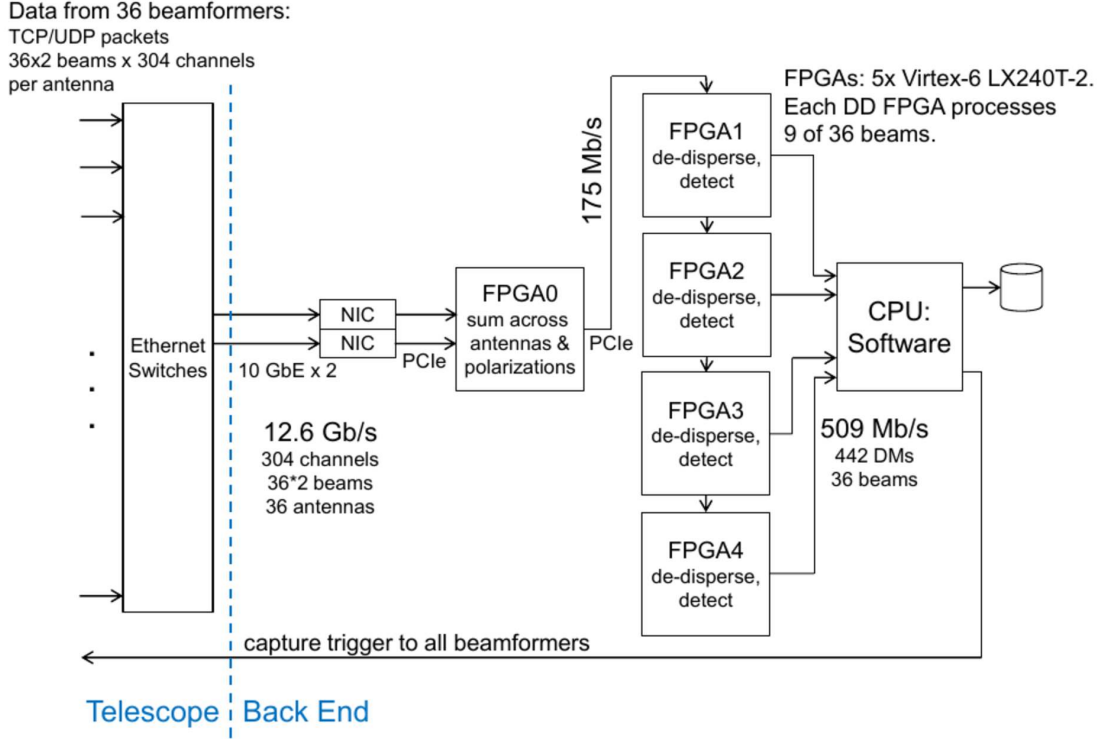


Figure 3.12: TARDIS: CRAFT fast transient backend implementation. Source: [D’Addario et al., 2013]

performance. By comparison, the test benchmark presented in the previous section managed 138×10^9 operations per seconds on two NVIDIA GPUs. These approximations are based on the direct-dedispersion method, however [D’Addario et al., 2013] use a different version which joins contiguous samples together if the dispersion lines span multiple time bins, thus increasing the operation cost by a small factor. Even so, these figures suggest that GPUs are more suited for real-time transient pipelines than FPGAs. The latter would also suffer from lack of on-board memory available when faster sampling times and high DM values are required. An additional advantage in favour of GPUs, when compared to FPGAs, is the ease with which new algorithms can be implemented and attached to existing pipelines. On the other hand, although a high end Virtex 6 costs more than a GPU, the running costs of an FPGA-based system are less, due to lower power requirements. This can be countered by appropriate upgrade paths throughout the lifetime of the system, as newer generation GPUs will generally have a higher performance per watt ratio, and upgrading the devices would yield savings in the long run.

3.10 Conclusion

Fast transient detection pipelines require considerable computational resources, and several processing platforms are being investigated to suite these requirements. In this chapter we propose a GPU-based solution, moulded around a custom GPU framework, whereby data input and buffering is performed by the CPU, after which this data are copied to GPU memory where compute-intensive tasks are performed. It is capable of processing multiple independent beams in parallel across as many GPUs as are attached to the host. RFI mitigation through bandpass correction and thresholding removes high power spatial and temporal terrestrial signals which would result in a high number of false detections. The dedispersion kernel presented in chapter 2 is integrated within this pipeline as well. After an optional post-processing stage, the dedispersed time series from each beam are copied to host memory where event detection through thresholding and density-based clustering is performed. The DM-S/N curve of each cluster candidate is then compared to a fitted model which filters out RFI-induced clusters and sends a trigger to the data persistence module, which quantises and writes the input buffer to disk containing the event, together with cluster parameters.

The accuracy of the RFI mitigation and event detection stage was also examined. Simulated data sets containing RFI and dispersed pulses were passed through the pipeline, the output of which was examined. Narrow pulses, of the order of 2 to 10 samples wide, were only fully detected for a pulse S/N of 4.0 or greater, while wider pulses were detected for pulse S/N as low as 2.0. Most of the simulated RFI signals did pass through the RFI mitigation stage, however 100% of the broadband signals were correctly classified as RFI, whilst a single cluster caused by a narrowband burst was incorrectly classified as an astrophysical event.

CHAPTER 4

DEPLOYMENT AT THE MEDICINA BEST-II ARRAY

In the previous chapter we have described in detail a generic transient detection pipeline and demonstrated the applicability of GPUs for deploying such a system to the backend of radio telescopes. In this chapter we enhance this pipeline with real-time streaming capabilities and attach it to the digital backend of the Basic Element for SKA Training II (BEST-II) array in Medicina, Italy. This serves as an excellent test-bed for benchmarking and conducting test observations. We begin by introducing BEST-II and the ROACH-based digital backend, and then move on to describe the deployment setup for our transient detection pipeline and present the results from various test observations, most notably of PSR B0329+54.

4.1 The BEST-II SKA pathfinder

The Basic Element for SKA Training II (BEST-II) [Montebugnoli et al., 2009] is a subset of the Northern Cross cylindrical array (figure 4.1) at the Medicina observatory near Bologna, Italy. The array is composed of eight East-West oriented cylindrical concentrators, each with 64 dipole receivers spaced such that the cylinder focal line is critically sampled at 408 MHz. Signals from the 64 dipoles are combined in groups of 16 using analogue circuitry, resulting in four analogue channels per cylinder. This results in a total of 32 effective receiving elements positioned regularly on a 4 x 8 grid, as shown in figure 4.1a. The top level specifications of the BEST-II array, which provides $\sim 1^\circ$ resolution over ~ 38 square degrees FoV, are summarised in table 4.1.

Such a receiver configuration leads to a set of highly redundant baselines,

BEST-II Specifications	
Total Collecting Area	1411.2 m ²
Total Effective Area	1001.95 m ²
Receiver Temperature	51 K
System Temperature	86 K
$A_{\text{eff}}/T_{\text{sys}}$	11.65 m ² /K
Longest Baseline (N-S)	70 m
Longest Baseline (E-W)	17 m
RF band	400 - 416 MHz
Total analogue channels	32
Primary FoV (Dec, 408 MHz)	5.7°
Primary FoV (RA, 408 MHz)	6.6°
Synthesised FoV (Dec, 408 MHz, at zenith)	0.52°
Synthesised FoV (RA, 408 MHz)	1.73°

Table 4.1: The top-level specifications of the BEST-2 array. Source: [Hickish, 2013]

which allows efficient spatial processing using Fourier techniques. In the case of BEST-II, the array possesses 496 possible antenna pairings (excluding pairings of an antenna with itself), forming 52 unique baselines, which is an ideal test-bed for testing and deploying a Direct Imaging Correlator. The digital backed, developed and deployed by Jack Hickish [Hickish, 2013] and Griffin Foster [Foster, 2013] from the University of Oxford, is based on three Reconfigurable Open Architecture Computing Hardware (ROACH¹) processing platforms and a number of software-

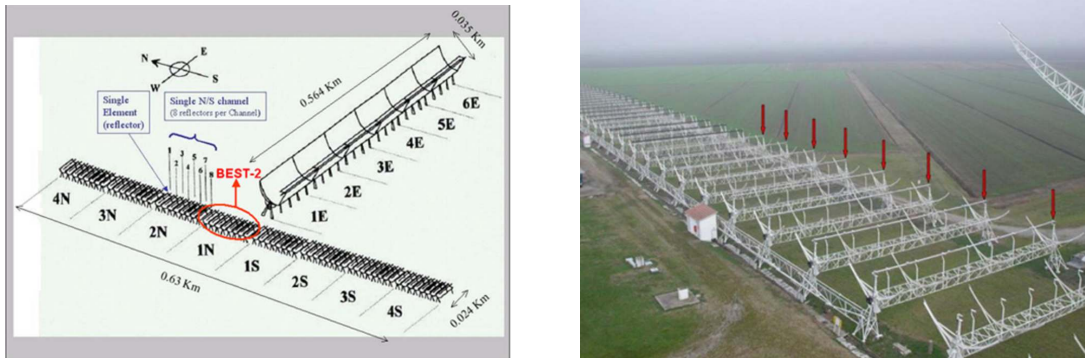


Figure 4.1: (a) A schematic of the Northern Cross Telescope, indicating the subset of eight reflectors of the N-S arm which form the BEST-II array. (b) A photo of the BEST-II array.

¹ <https://casper.berkeley.edu/wiki/ROACH>

based processing nodes. The overall topology and architecture of this system is shown in figure 4.2. The three ROACH boards are referred to as the:

“F”-engine for frequency transform, which is responsible for digitisation, channelisation of the processed 20 MHz bandwidth into 1024 frequency channels, and transmission of coarsely quantised antenna signals to downstream processing boards. All 32 analogue streams are digitised on a single processing node using a custom 64ADCx64-12 board². A 4-tap Hann-windowed polyphase filter bank is used for channelisation.

“S”-engine for spatial transform, which is responsible for formation of electric-field and total power beams on the sky by spatial Fourier transform and SPEAD packetisation for 10GbE streaming to the time domain processing server. The current implementation allows for the arbitrary selection of 8 beams from the generated 128-beam grid for output as 16-bit complex-valued words.

“X”-engine for cross-multiplication, which performs cross multiplication and accumulation of antenna signals as well as SPEAD packetisation of the generated visibility matrices and streaming to the visibility storage server for offline imaging. This is also used to calibrate the S-engine.

Data is passed between ROACH boards using the Ten Gigabit Attachment Unit Interface (XAUI) over a copper cable connection, which is a lightweight, bidirectional, point-to-point transmission protocol. Downstream processing of time-domain beam data is accomplished by using a Linux-based server, which hosts 2 NVIDIA GPUs running our transient detection pipeline. The 8 beams selected for output from the S-engine are packed as 16-bit complex values and transmitted over 10GigE using a custom Streaming Protocol for Exchanging Astronomical Data (SPEAD³) [Manley et al., 2012] packet format. SPEAD is a flexible, self-describing, lightweight application-level datastream format. It is designed to provide a standard output format for radio astronomy instruments outputting UDP streams. The output rate of a single beam from the beamformer is 640 Mbps excluding packet headers, which is calculated using $D = C \times T \times W$, where C is the number of frequency channels, T is the number of time samples per second and W is the word length. In our case, $C = 1024$, $T = 19531.25$ (20 MHz processable bandwidth channelised into 1024 channels) and $W = 32$ bits (16 bits for each

² <https://casper.berkeley.edu/wiki/64ADCx64-12>

³ <https://casper.berkeley.edu/wiki/SPEAD>

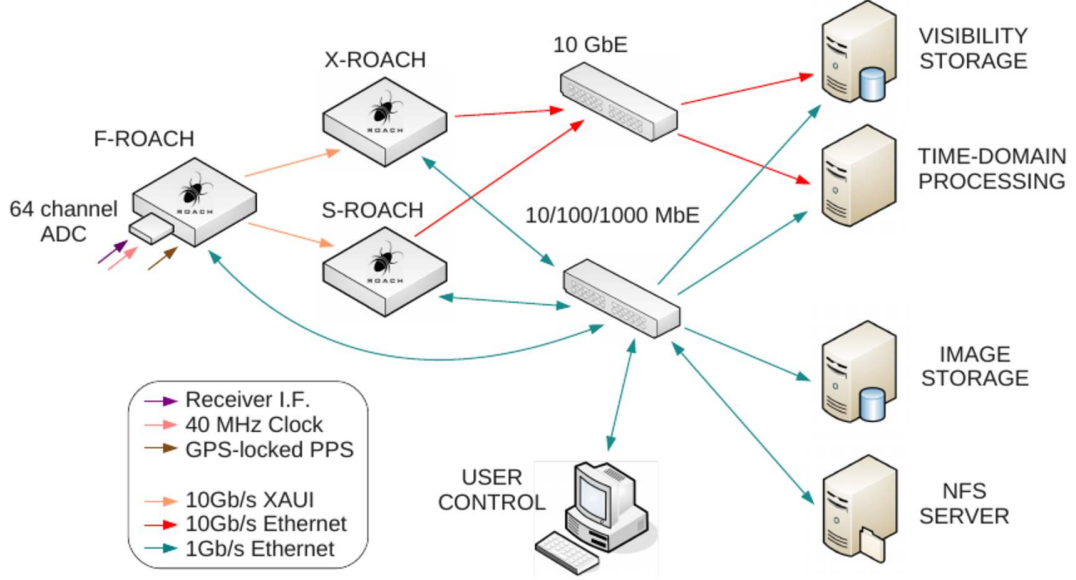


Figure 4.2: The top level architecture of the BEST-II backend. 10Gb and 1Gb links are used to transport data from the ROACHes to CPU and GPU nodes. 10Gb/s XAUI is used for ROACH-ROACH connections. “F”, “X” and “S” ROACH boards are responsible for channelization, cross correlation and spatial FFT respectively. The roles performed by any of the CPU-based nodes need not be split amongst physically distinct hosts. Source: [Hickish, 2013, figure 4.6]

complex component). A total output bandwidth of 5.12 Gbps is required to send out all 8 beams, which is manageable over a single 10GigE link.

4.2 BEST-II transient detection pipeline

In chapter 3 we have described in detail a generic GPU-based transient detection pipeline. Here we enhance this with real-time capabilities, which takes the form of a high-speed, optimised packet receiver. The high-level architecture of this pipeline is depicted in figure 4.3, which is split into three main processing stages: the data reception and buffering stage, the GPU-based processing pipeline stage and the post-processing stage. Beams are processed independently across multiple GPUs, and a CPU thread is associated with each beam, where multiple beams can reside on a single GPU.

Packet reception, interpretation and buffering is performed on the CPU, which then forwards the data to the attached GPUs where it passes through sev-

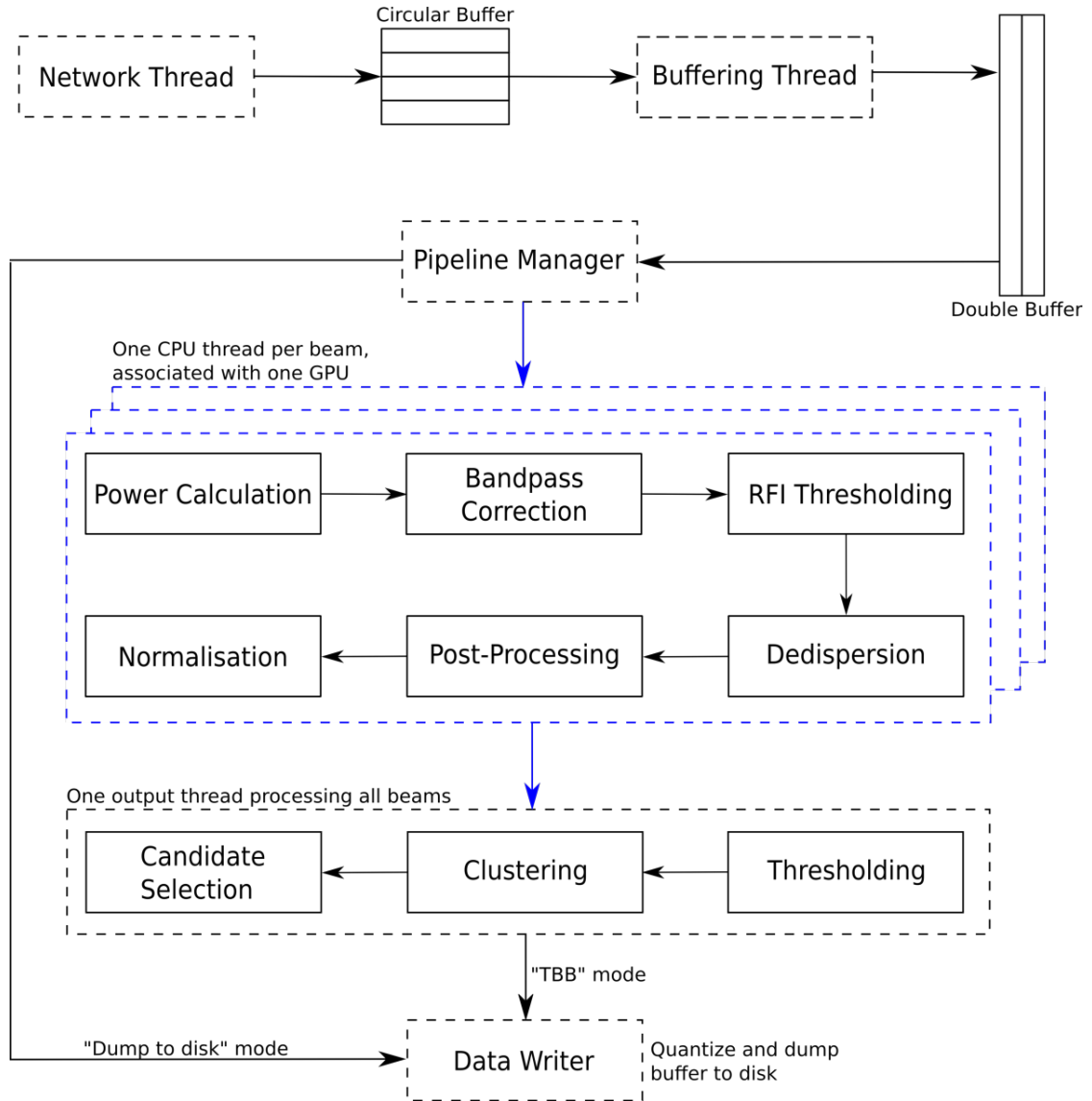


Figure 4.3: Overview of the system and pipeline architecture. Boxes in dashed lines represent distinct CPU threads, whilst dashed blue overlays represent CPU threads which use GPUs for processing. Blue arrows represent data transfers to/from GPU memory.

eral processing stages including: power calculation, bandpass correction, RFI clipping, dedispersion and optional post-processing and normalisation, after which the dedispersed time-series are copied back to CPU memory and passed through a detection stage where it is thresholded, clustered and classified. Any data-points belonging to interesting clusters are written to disk, together with the unprocessed data buffer after being quantised to 8 or 4 bits depending on whether the data has been converted to a channelised power-series in the receiver thread. There is

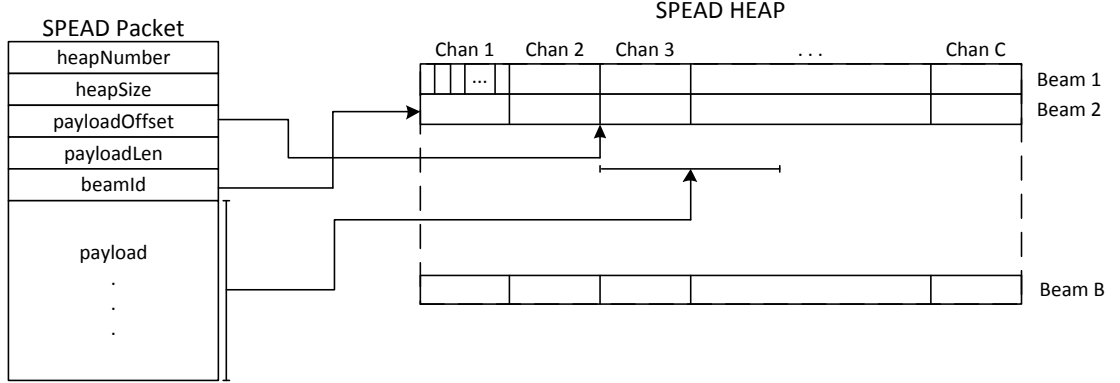


Figure 4.4: SPEAD packet format and heap data organisation used to send data between the S-engine and GPU server. A heap contains a number of spectra for all the beams being processed, for all the frequency channels. This is split up across multiple packets which contain time samples from a subset of the frequency channels, for a single beam. The packet header defines where the data should be located within the heap, as well as information to be able to identify heaps and extract timing information.

also the possibility of writing the entire data stream to disk after passing through an encoding and quantisation stage, provided that the disk drives can manage the reduced data

4.2.1 Packet Receiver

The S-engine packetises the channelised beamformed data using a custom SPEAD packet format designed to reduce the overhead of heap generation and data movement on the receiver side. A heap consists of a time-slice containing several spectra (composed of 16-bit complex values sampled for all channels) from multiple beams, organised in beam/channel/time order, which maps directly to the data organisation in GPU memory, thus considerably reducing the overhead for memory re-arrangement. Figure 4.4 describes in further detail the data organisation of a heap and how this maps to the packet format. The UDP transmission protocol is used to send the packet stream, since it is lightweight and the probability of losing packets, or receiving them out of order, is very small and can be easily handled using simple error checking routines on the receiver side, where heap buffers provide a small time window during which an out-of-order packet can still be processed correctly. The heap size is set by the digital backend.

Care needs to be taken when designing receiver codes for these types of data streams. A saturated 10GigE link can achieve a total data rate of ~ 1.25 GB/s, which needs to be processed in real-time, usually by a single CPU thread, with minimal resource consumption on the host system. Some of the major factors which affect receiver performance include:

Packet Size Every received packet needs to pass through the protocol stack before reaching its destination and is ready for processing, where the respective headers are stripped, checked and processed. This induces a processing overhead which can become severe when the payload size to frame header ratio is small. Ideally the packet size would be the maximum possible allowed by the transmitting backend and network devices between the two end points. The S-engine transmits packets containing a payload of 4096 bytes, resulting in 156,250 packets per second.

Data Movement Even if the incoming packets are read once from the network socket and then discarded there's an automatic penalty of data movement between kernel space and user space, together with the associated context switch required to access kernel memory space. Once in user space, care needs to be taken with the buffering scheme used, as large memory access strides will result in a large number of cache misses and performance degradation.

System Calls System calls require context switches, and therefore must be reduced as much as possible. At least one is usually required to read a packet for the kernel UDP buffer, and sometimes polling mechanisms are employed to wait for incoming data, resulting in at least two system calls per packet, which is not the ideal case.

The latter two factors can be greatly alleviated by careful design of the receive code. In our implementation we make use of the PACKET⁴ socket interface, which provides a size configurable circular buffer, mapped to user space that can be used to either send or receive packets. This buffer is partitioned into blocks, each containing number of frames, where each frame is a placeholder for a network packet. With this mechanism, the user code simply waits for a frame to become available in this ring buffer (essentially waiting for a DMA transfer from the network adapter's internal memory to the circular buffer) and most of the time there

⁴https://www.kernel.org/doc/Documentation/networking/packet_mmap.txt

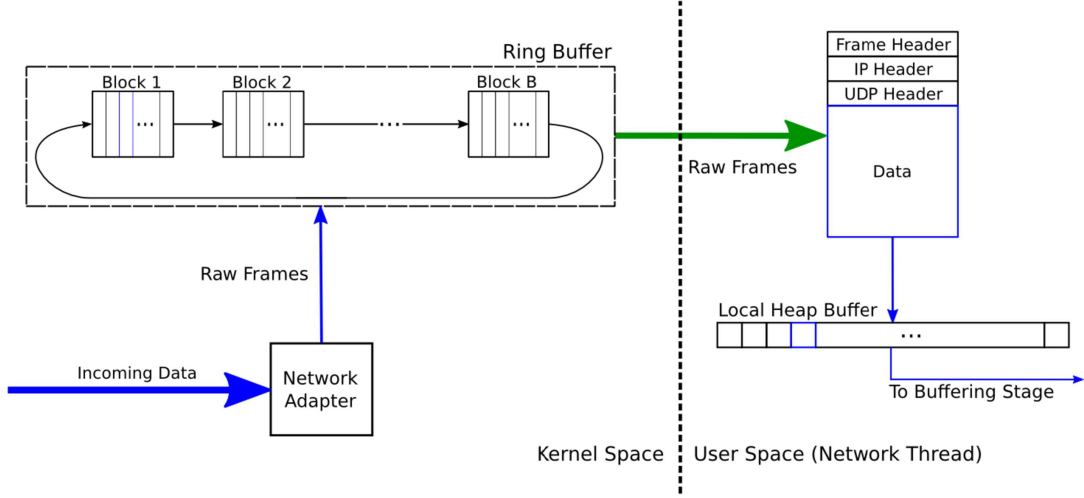


Figure 4.5: Packet receiver schematic, which uses PACKET sockets and memory mapping between kernel and user space for high speed packet reception. Incoming packets follow the arrows in blue, while the green arrow depicts the zero-copy mechanism between the two memory spaces.

is no need to issue system calls since the frames are read directly from user memory. Using a shared buffer between the kernel and the user also minimises packet copies, and can be thought of as zero-copy memory access. The major downside of the scheme is higher implementation complexity in the user code, which has to strip down the protocol headers itself. The internal workings of this scheme is illustrated in figure 4.5

The incoming UDP stream is received and buffered for processing using two CPU threads, the network thread and the buffering thread. When a new UDP packet is received, the network thread reads and interprets the SPEAD header, which defines the heap it belongs to and where its payload should be placed within the heap, after which it is copied to the specified offset. This requires a heap buffer which is kept local to the network thread until the entire heap has been received. A circular heap buffer is shared between the network and buffering threads, allowing their operations to overlap and minimise locking overheads. When a heap is fully read, the buffering thread is notified and a new heap buffer is returned to the network thread, which starts receiving the next heap. If a packet from the next heap is received before the current heap is fully populated, then all missing packets are considered dropped, thus resulting in zeroed-out “gaps” in the heap, which are then handled by the RFI thresholding stage. Increasing the number of slots in the circular buffer reduces the probability that the network thread is kept waiting

for the buffering thread to free up slots, which might happen when CPU-intensive tasks are scheduled on the same core as the buffering thread.

The buffering thread’s main function is to create larger data buffers to be copied directly to GPU memory, composed of multiple heaps. To overlap the creation of these buffers with GPU execution, a double-buffering system is used. Heaps from the circular buffer are separated into chunks containing a time-slice from a single channel per beam, and these are copied to their respective locations within the GPU buffer. When this is fully populated, the main pipeline thread is notified and the pipeline is advanced by one iteration after all the GPU-based processing has finished for the previous one.

Mapping the network and buffering threads to specific CPUs and cores can have an effect on overall system performance. On a multi-CPU system, specific Interrupt Requests (IRQs) are generally handled by a subset of the available cores, the mapping of which is determined by a hexadecimal bit mask (on Linux systems). In cases where PCIe slots are controlled by multiple I/O chips, assigning the IRQ related to the 10GigE card to a core within the physically closest CPU will reduce latency and overhead. The network thread’s affinity can then be assigned to the same core. An additional optimisation option is to set the buffering thread’s affinity to a core residing on the same CPU in order to avoid false sharing situations among cache lines residing on different CPUs, which can result in performance degradation, and thus assures that cache lines containing circular buffer values are not invalidated. The actual performance gain achievable with these fine tunings is system dependent.

4.2.2 Data Interaction Tool

The transient detection pipeline can persist data to disk in several formats, with varying word lengths, depending on the stage at which the triggering occurred. A custom tool for data interaction was required to check the correctness of the pipeline, as well as the data streamed by the S-engine in order to make sure the digital backend was properly initialised. This tool would also serve the purpose of “empirical parameter setting”, in which several threshold factors needed to configure the pipeline, including the RFI thresholds, can be tested and determined. Due to the potentially large data files which would need to be processed, and the processing steps required for this visualisation, including decoding the μ -law encoded data, this tool was written in C++, whilst the user interface was designed and

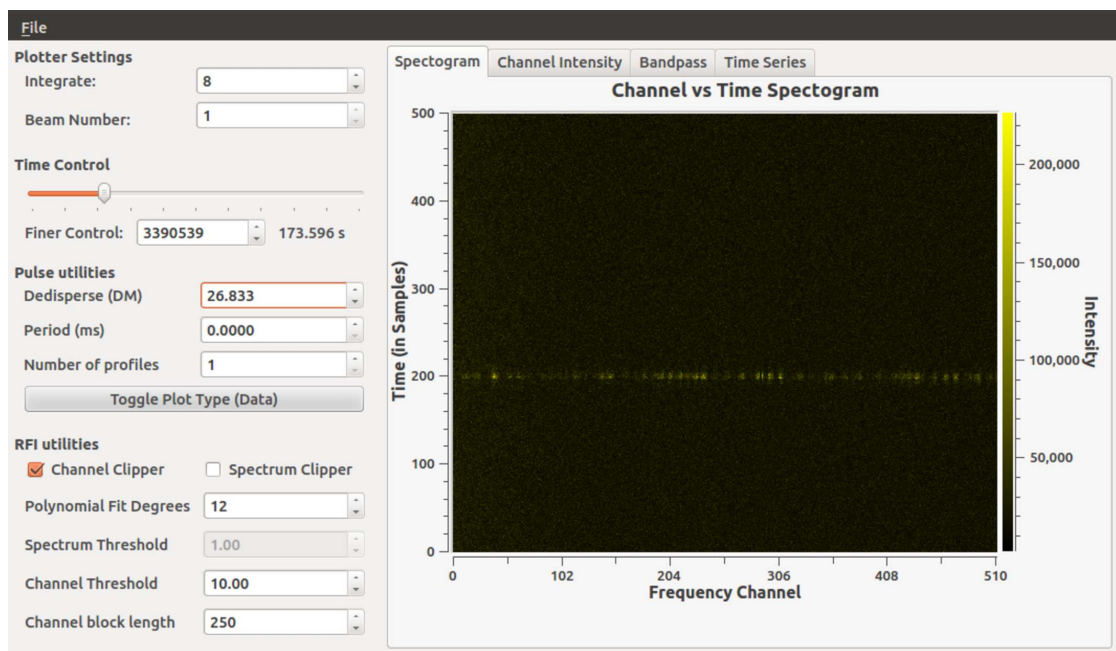


Figure 4.6: Data interaction tool written in C++ and QT which can process data files written to disk by the transient detection pipeline. The waterfall plot shows a dedispersed pulse from PSR B0329+54, integrated by a factor of 8.

developed with QT⁵. OpenMP was used to accelerate compute-intensive features. This front-end of this tool is shown in figure 4.6.

A typical use case for this tool involves loading a persisted data buffer, or a SIGPROC-style [Lorimer, 2006] filterbank data file, and decoding it, separating the beams and generating a different filterbank file per beam. Successive data buffers can also be combined into single observation files in this stage as well. This operation is parallelised across multiple beams. Once an observation file has been generated, the visualisation parameters are reset and the first group of spectra are plotted. The user can then use any of the available features to interact with the data:

- Analyse the data using various plot types, including: waterfall plot, frequency channel intensity plot, logarithmic bandpass plot and the time series plot (with frequency channels summed for each spectrum). These plots can also be saved to disk.
- Use the time control features to instantly analyse any part of the file, even if this involves seeking through gigabytes of data. Consecutive time spectra

⁵ <http://qt-project.org/>

can also be combined, or integrated.

- Use transient and pulsar-related features such as dedispersion and folding.
- Apply the RFI thresholding techniques described in section 3.3, with an optional user-defined channel mask.
- Export segments of the buffer, including folded profiles, as separate filterbank files to disk.

4.3 BEST-II Observation Results

The transient detection pipeline was deployed at the BEST-II array, attached via a 10GbE link to the S-engine. Initial tests indicated that the full 20 MHz band contained significant narrowband RFI, with 2 MHz at both edges of the band lacking any detectable flux since they lie outside of the BEST-II RF band. For this reason, the S-engine was configured to discard half of the band, using only 10 MHz for the rest of the observations, from 413.9 MHz to 403.9 MHz. Figure 4.7 shows the full bandpass, with the selected subband highlighted, as well as a waterfall plot of a single dispersed pulse from PSR B0329+54, for which the noisy channels were masked. An on-pulse, frequency-dependent, instrumental effect is also clearly visible in this plot, the source of which is still unknown. This results in a loss in S/N of any detected signals.

Since the BEST-II array is located next to an urban area, RFI is frequent and strong. Figure 4.8 shows two examples of both narrowband and broadband RFI events which occurred during our test observations. The efficacy of the RFI thresholding stage is also demonstrated, where both RFI events were detected and mitigated. Plots (c, d) also contain a transient signal originating from PSR B0329+54 which was being observed during the event. The transient signal was not affected by the RFI thresholding stage.

The S-engine creates a 2D grid of synthesised beams within the primary beam, out of which eight can be selected for output. The primary beams and centres of each synthesised beam, together with their FWHM, are shown in figure 4.9. Beams get wider further from the zenith, which for BEST-II is at 44.52° . These beams were chosen to create a “strip” along the E-W direction (along RA) so that pointing towards a transient source would result in it transiting across multiple beams, which is useful for testing the digital beamformer, the pipeline

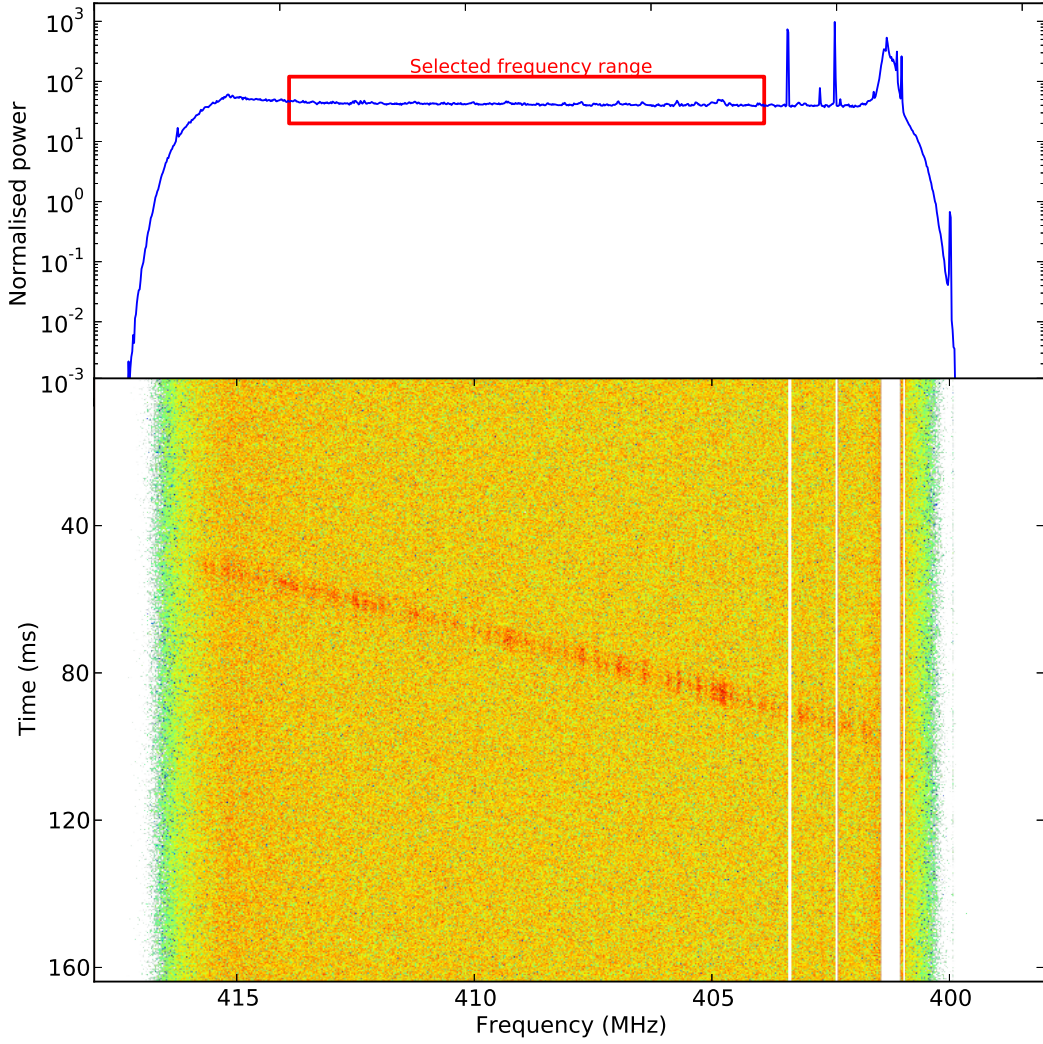


Figure 4.7: Top plot shows the full-band BEST-II bandpass, 10 MHz of which were selected for further observations, depicted by the red border. Bottom plot is a full-band waterfall plot of a single dispersed pulse from PSR B0329+54. The colour map represents normalised, logarithmic power. Noisy channels were masked.

setup as well as the data transmission between the two. It should be noted that neighbouring beams overlap at $\sim 81\%$ of peak power, therefore S/N is reduced when a source lies between beam centres.

Several test observations were performed on known bright pulsars, especially PSR B0329+54, which is the brightest transient source that can be observed with BEST-II, located at RA 03:32:59.37 and DEC +54:34:43.57. The beam configuration for these observations is shown in figure 4.9, where the central row of beams is selected for output (shaded in red). The pipeline was used in “persistence mode”, where all the channelised complex-voltages from all the beams are quantised and

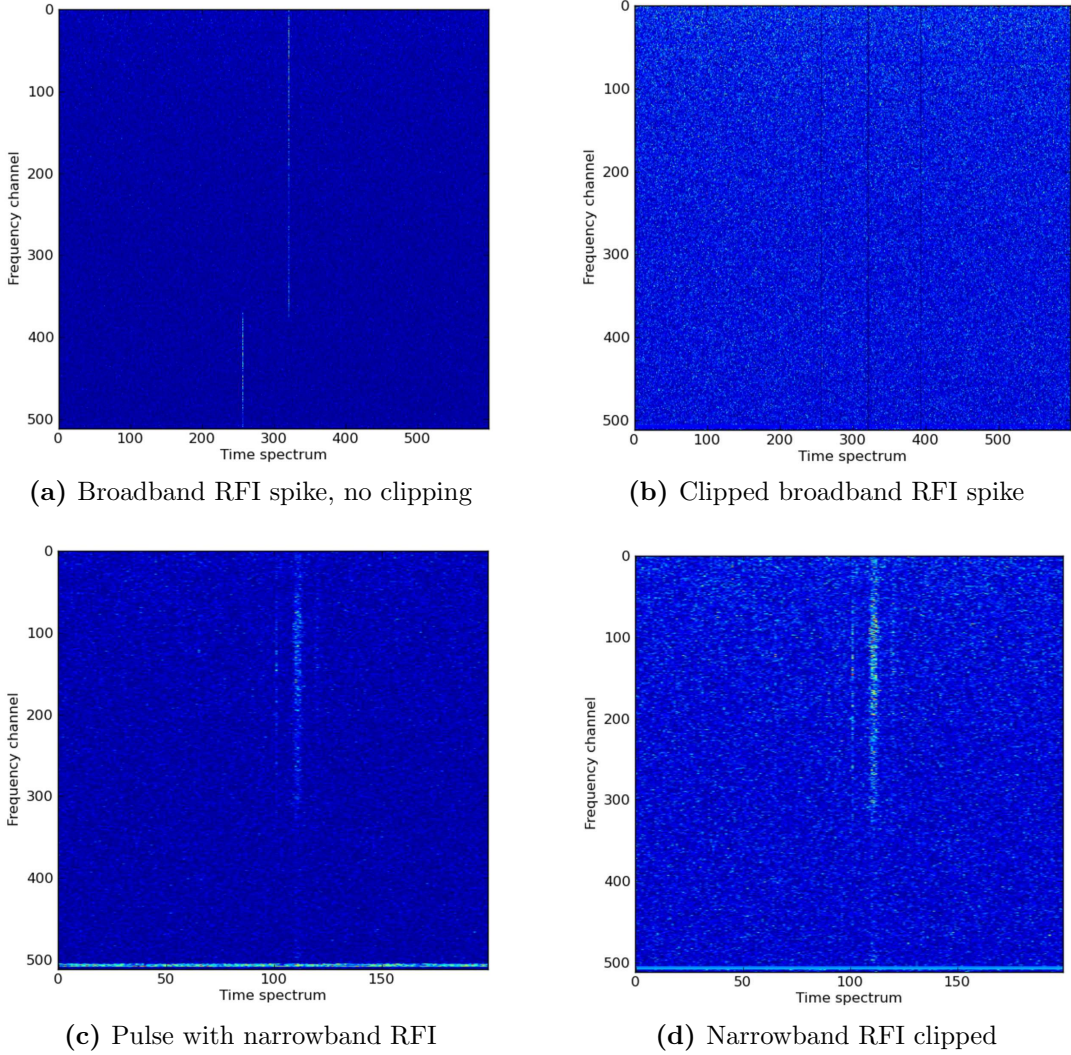


Figure 4.8: RFI events during a test observation using a 10 MHz bandwidth between 413 and 403 MHz. Plot (a) shows three broadband spikes affecting spectra at 260, 330 and 395, while (c) shows narrowband RFI affecting frequency channels 504 to 508. Plots (c, d) also contain a transient signal originating from pulsar PSR B0329+56 which was being observed during the event. For both cases all thresholding stages were enabled and had the same threshold parameters. The power intensity values are arbitrary.

persisted to disk. The integrated pulse profiles were then generated for each beam using 50 profiles. The differences between the profiles can be attributed to RFI events which occurred during the observation, sometimes resulting in a recalculation of the quantisation factors (this only happens when an extremely powerful RFI event occurs, as was the case during this observation).

The integrated pulse profiles for PSR B0329+54 is shown in figure 4.10, generated by folding 200 profiles offline with raw observation data persisted to

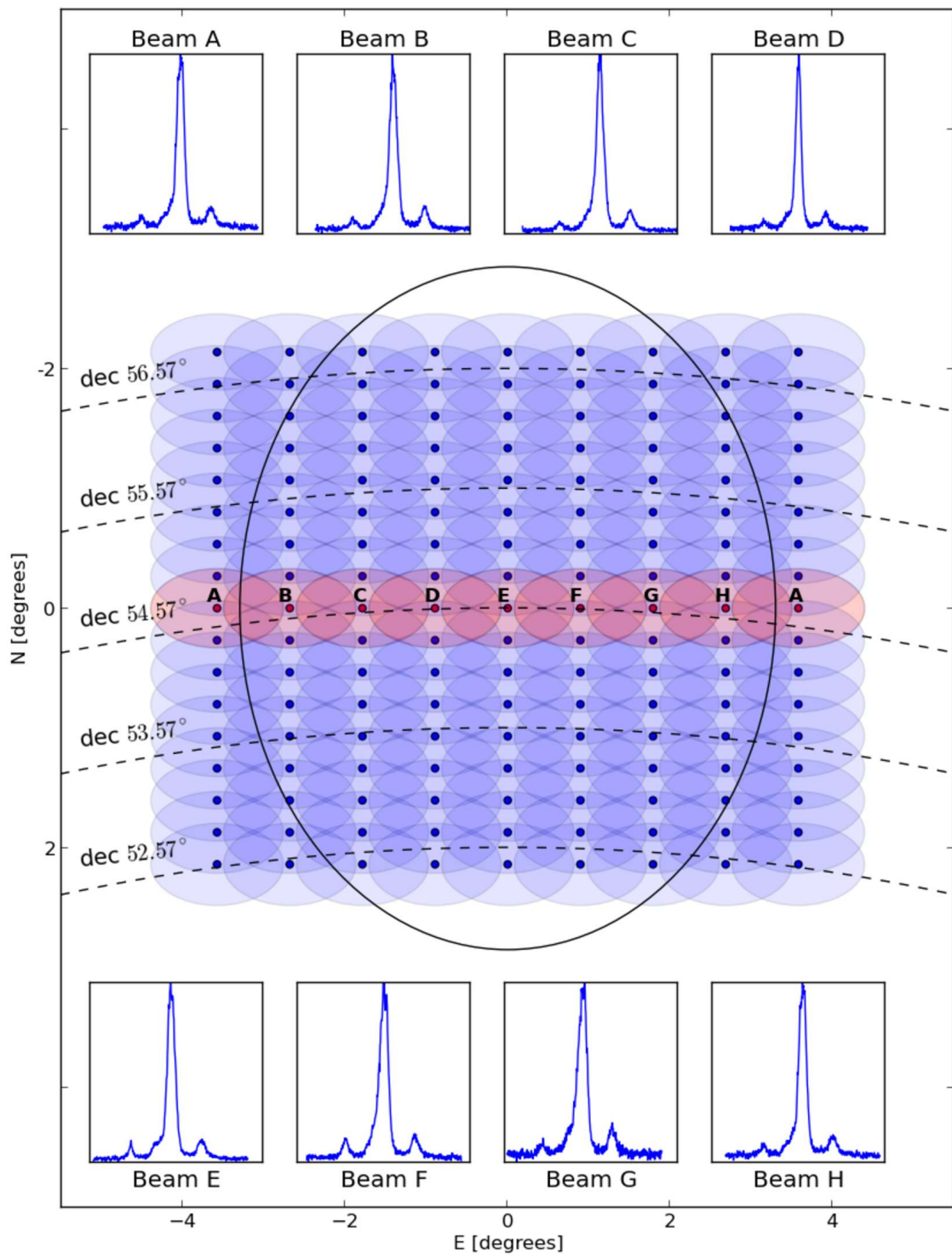


Figure 4.9: S-engine beam configuration, also showing a transiting test with PSR B0329+54 during a 1800 s observation. The integrated pulse profile for each beam is also shown, generated by folding 50 profiles.

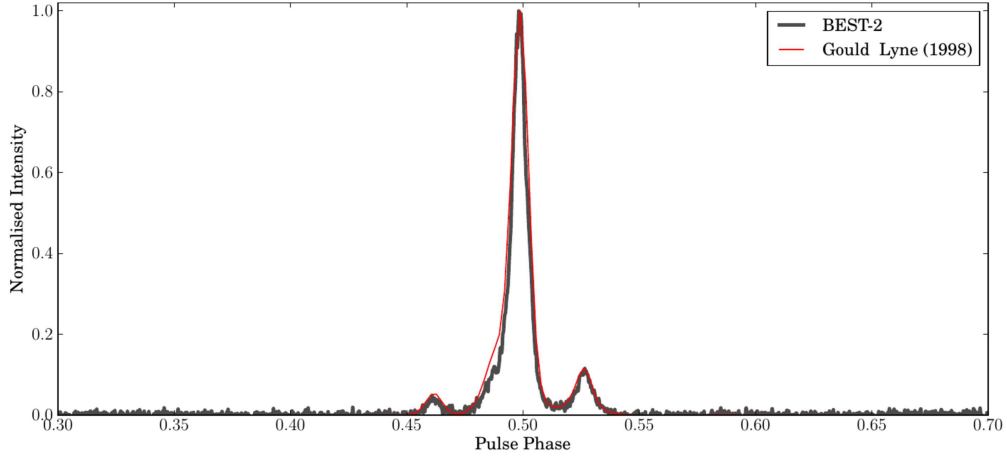


Figure 4.10: Integrated pulse profiles of PSR B0329+54 and consisting of 200 profiles overlaid over a profile obtained using the Lovell telescope at 408 MHz [Gould and Lyne, 1998]. (Credit: Jack Hickish)

disk from a single beam and overlaid over a profile obtained using the Lovell telescope at 408 MHz [Gould and Lyne, 1998].

Figure 4.11 represents the output of the pipeline during an observation of PSR B0329+54 for a single beam. The pulsar enters the beam at ~ 100 s, with pulses getting stronger as it moves towards the beam’s centre. Several RFI events were also detected, the most noticeable of them being three broadband events (the large detections at $DM \approx 0$) and a bright narrowband event at around 215 s which was detected across the entire DM range. Figure 4.12 provides visual snapshot of four clusters during the classification stage, with two clusters originating from pulse detections from B0329+54 and two additional clusters attributed to RFI. Both these RFI clusters were correctly filtered.

4.4 Conclusion

We have enhanced the GPU-based transient detection pipeline described in the previous chapter with real-time capabilities, consisting primarily of high-speed stream processing functionality capable of processing a 5.12 Gb/s SPEAD stream. We have also described the deployment of this system on the Medicina BEST-II array, where a ROACH-based digital backend performs all the required pre-processing prior to transmitting beamformed data to the transient detection system. Several test observations were conducted, especially on PSR B0329+54, which is the brightest pulsar observable by BEST-II. Through these observations appropriate

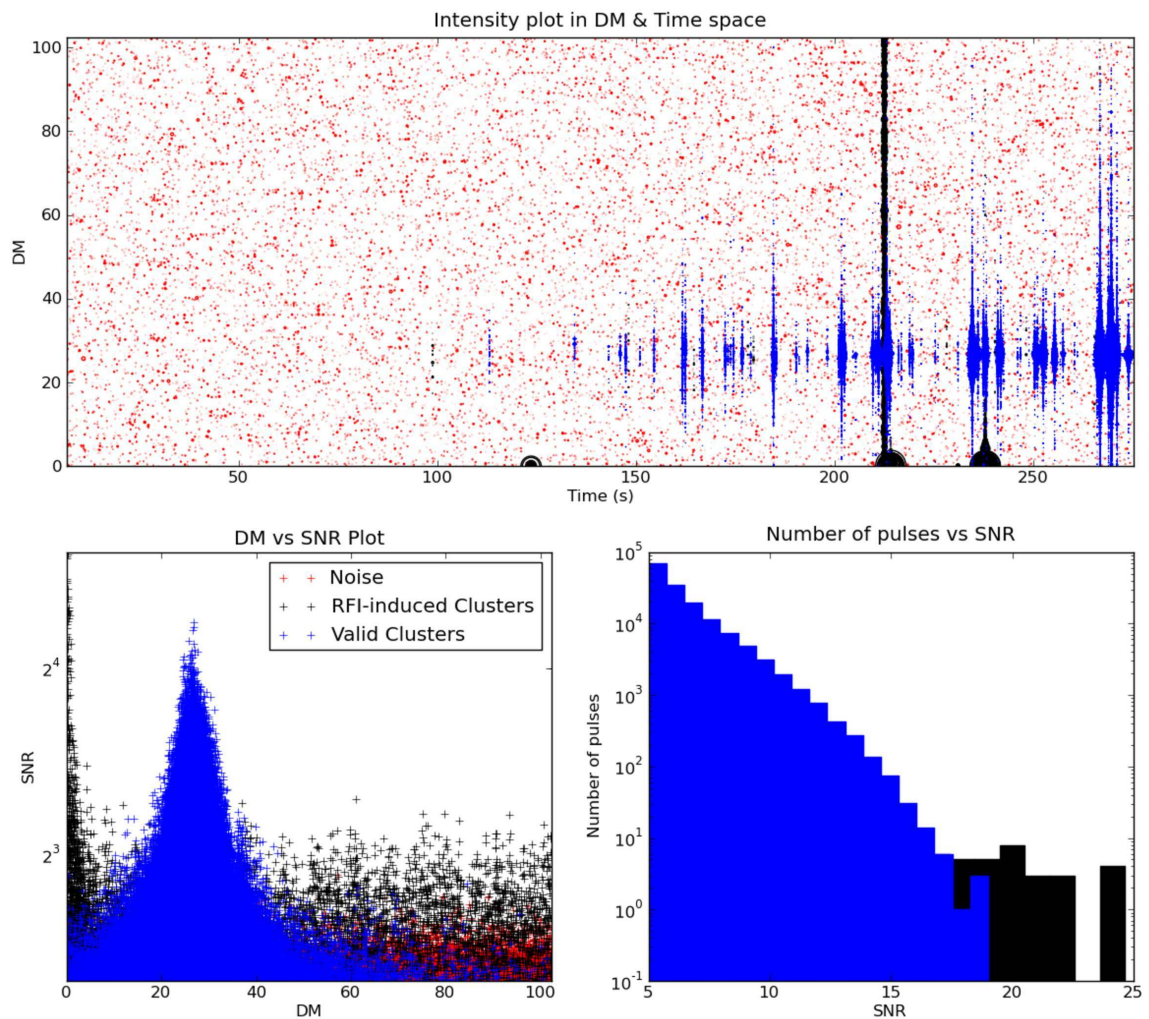
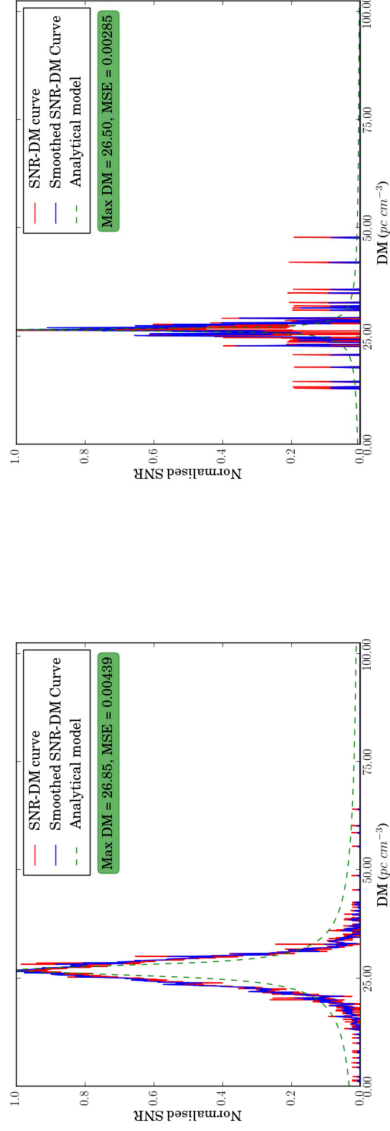


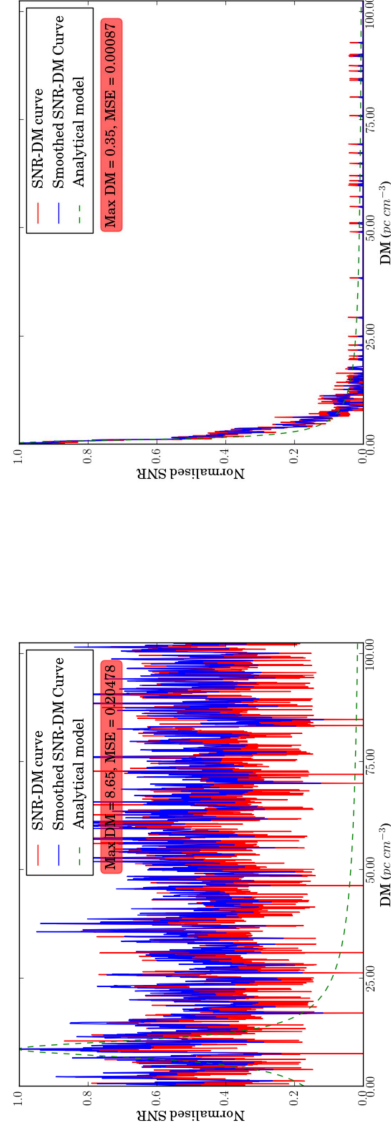
Figure 4.11: A ~ 280 s observation of pulsar PSR B0329+54, which enters the beam at around 100 s with the pulse S/N increasing as the pulsar moves towards the centre of the beam. This plot shows the output generated by the clustering and candidate selection stages, partitioning the data into noise (red), RFI-induced clusters (black) and selected candidates (blue). Some of these clusters are also shown in figure 4.12.

threshold parameters were set to various stages in the pipeline, leading to the automatic classification of pulses originating from B0329+54 from RFI-induced events, serving as an online test case for the pipeline.

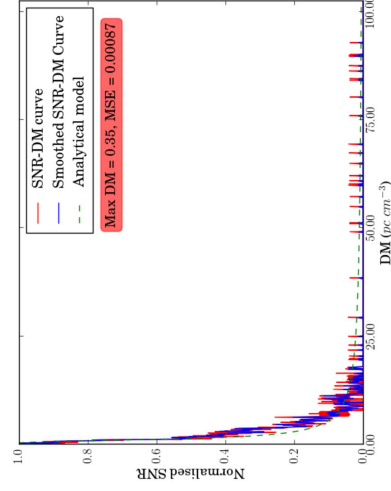


(a) High S/N pulse from B0329+54

(b) Low S/N pulse from B0329+54



(c) High-intensity narrowband RFI



(d) Broadband RFI

Figure 4.12: Examples of clusters during the candidate selection stage. The first two plots belong to pulses from B0329+54 whilst the rest are due to RFI. Plot c represents the black vertical cluster at around 215 s in figure 4.11, whilst plot d is the low DM, high-S/N detection at 240 s.

CHAPTER 5

GPU-BASED BEAMFORMING FOR TRANSIENT DISCOVERY

In the previous chapters we have described the design, implementation and deployment of a real-time, GPU-based transient detection pipeline, where the host servers receive beamformed antenna signals for processing. The design is linearly scalable with increasing number of beams, which can be parallelised across multiple GPUs and servers. However, without appropriate feedback mechanisms to the beamformer, observations will be limited to the observational parameters set during initialisation. The beams' shape, distribution across the array's FoV, sensitivity and sky coverage depend on the beamformer's implementation and supported parameter range. This is especially limiting for FPGA-based designs, where simply changing the beamformer's mode (for example, from incoherent to coherent) would generally require a separate design to be loaded and configured. An alternative approach is to have the digital backend stream the raw antenna voltages to the host system, where beamforming is performed within the transient detection pipeline itself.

This setup offers several advantages. Switching between beamforming modes can be performed in real-time and with minimal overhead, with the possibility of generating a mixture of beam types. Such a system would be useful for on-the-fly localisation and tracking of transient candidates, whereby coherent beams can be generated whilst still using the default beam setup for full FoV scanning. Having the raw antenna voltages available on the host system also enables the possibility of generating an image of any detected candidates, where sources can be better localised. Performing this in real time allows immediate follow-up observations, however this requires accurate source extraction techniques in images, which are still being investigated, especially for slow transient surveys. Several authors also

investigated the benefits of sub-arrays for transient detection (for example, [Cordes, 2009, Macquart, 2011, Colegate and Clarke, 2011, Bhat et al., 2013]), where an array is partitioned into sub-arrays, each of which is incoherently beamformed and processed, with the benefit of being more resilient to localised RFI when using coincidence filtering techniques.

In this chapter we investigate the applicability of GPUs for beamforming purposes. We design and implement a standard, GPU-based, coherent beamformer which can generate multiple beams with arbitrary directionality. We also integrate this kernel with the transient detection pipeline described in chapter 3 and demonstrate the performance and flexibility of this system.

5.1 Beamforming

In an array with N elements a radiation wavefront originating from a particular direction will reach element n at time n_t . Adding signals from all these elements together without compensating for element-dependent delay is the process of incoherent beamforming. The maximum amplitude is achieved when the signals originate from a source perpendicular to the array, where they are highly correlated and add constructively. Alternatively, if the signals originate from a non-perpendicular direction they will arrive at different times, will be less correlated and result in a lower output amplitude. Incoherently combined arrays are referred to as incoherent arrays (IA). Coherent beamforming relies on the fact that for a given array configuration the relative delays between arrival times $t_{0..N-1}$ are a function of the direction of propagation of the incident wave. Artificial delays or phase shifts, in the time and frequency domain respectively, can be applied to the received signals from each antenna, causing them to add constructively when element signals are summed. This process is illustrated in figure 5.1 for a simple 3-element, one-directional array. Coherently combined arrays are referred to as phased arrays (PA).

After summing to form a beam towards the incident radiation with wavevector k_0 the response $B(k)$ of the beam to radiation with other wavevector directions $k(\theta, \phi)$ is given by summing the signals contributions from individual antennas. If each individual antenna in an array has a response to radiation given by $A_n(k)$, this is given by:

$$B(k) = \sum_{n=0}^{N-1} A_n(k) e^{i(k-k_0) \cdot r_n} \quad (5.1)$$

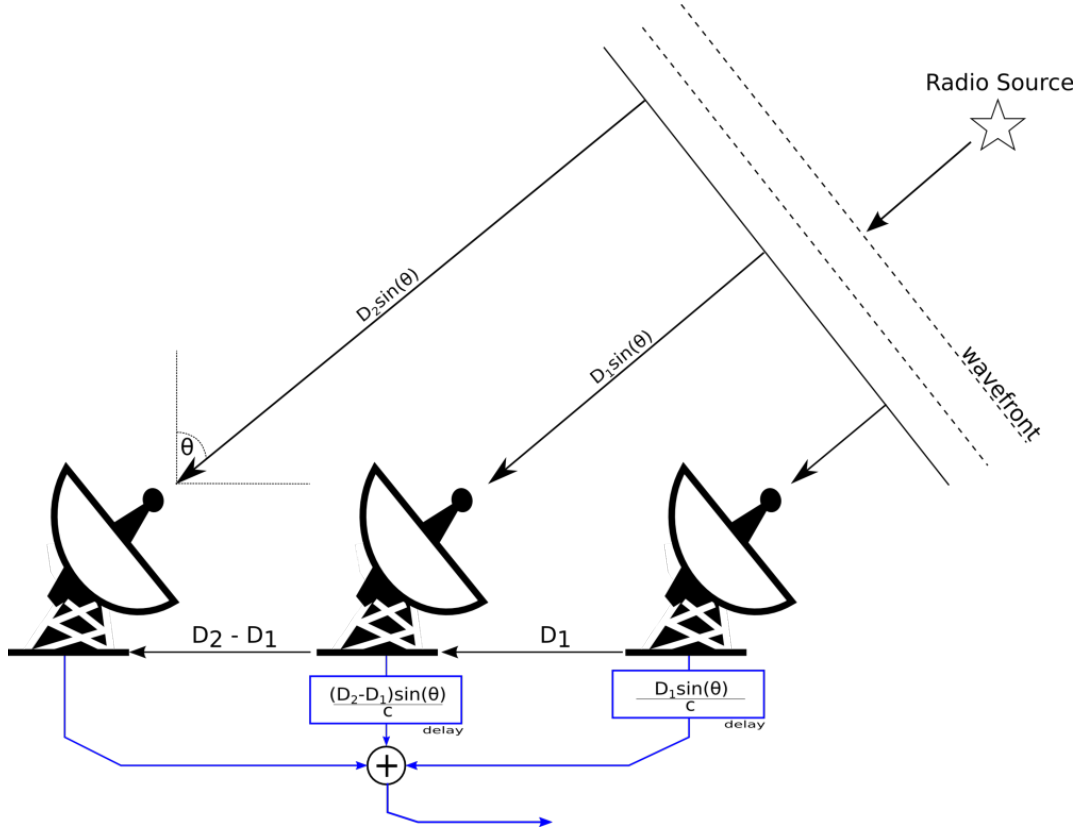


Figure 5.1: Schematic of a simple 1D, 3 antenna beamformer forming a beam in direction θ . Signals from each spatially separated antenna are delayed and summed such that the response is greatest in the desired direction. Figure adapted from [Hickish, 2013].

where r_n is the position vector of antenna n . This is equivalent to the Fourier transform of the contributions from receiving elements weighted by their individual antenna responses. In cases where the antennas have very similar response functions equation 5.1 reduces to the Fourier transform of the antenna distribution modulated by the response pattern of each antenna. In this case, the FoV of the synthesised beam is approximately given by λ/r_{\max} , where r_{\max} is the maximum separation of the antennas used to form the beam.

Since prior to beamforming each individual antenna in an array sees a portion of the sky determined by its response function $A_n(k)$, it is possible to effectively form beams in any direction where A_n is significantly non-zero. It is also possible to generate multiple beams by making copies of the input antenna signals and adding them with different phase weightings, thereby increasing the filling factor of an array's full FoV. Choosing the number of beams to form amounts to balancing the FoV observed by an array (which increases linearly with each beam added) with the

number of signals required to be processed. Each beam can be treated as if it is a signal from a single antenna with a response equal to the beam response, and so can be fed directly to backend detectors and used for time-domain observation, such as a cosmic transient event observations and transient surveys. The latter generally require a wide FoV and high sensitivity, however achieving this for larger arrays can be prohibitively expensive in signal processing costs since the number of required beams goes as $(D/d)^2$, where D is the physical extent of the array and d is the size of an individual element. Alternative strategies include sacrificing sensitivity to achieve maximal FoV by using IA mode, or partitioning the array into multiple PAs and processing them independently.

5.2 Beamforming implementation on GPUs

In this section we describe a standalone GPU implementation of a coherent beamformer which is capable of generating multiple beams concurrently. Performing this in real-time poses several challenges. First of all the digitised antenna voltage streams need to be transported to the backend server, which has to cope with the generally high data rates. Network links and devices can achieve very high transfer rates, however in order to offload processing to a GPU all this data needs to be transferred through PCIe connections to GPU memory, which have limited peak bandwidth. This can create an upper bound on the number of beams which can be generated. Also, depending on the representation scheme used for the generated beams, the amount of available GPU memory might limit the level of parallelism achievable. These factors will need to be considered when a beamforming kernel is deployed within a real-time pipeline, and will be discussed in greater detail shortly.

Ignoring the weight calculation scheme used to point each beam, the computational complexity for a generic coherent multi-beam beamformer is $\mathcal{O}(N_b N_f N_a N_t)$ where N_b is the number of synthesised beams, N_t is the number of time samples, N_f is the number of frequency channels and N_a is the number of antennas. The beamforming process is essentially a matrix-vector multiplication, which involves multiplying a vector of N_a antennas by an $N_b \times N_a$ matrix of coefficients to form N_b beams, per frequency channel. This incurs a computational cost of N_a complex multiply accumulates per beam and frequency channel.

This algorithm is trivially parallelisable across the frequency, time and beam dimensions. A simple and naïve implementation would have a single thread combine all the antennas, applying appropriate weights, for a single (f, t, b) triplet.

This would require $2N_a$ global memory requests for every thread, since N_a antenna values and N_a complex weights are needed, with an instruction count of $8N_a - 1$, resulting in a flop to data request ratio of 4. Due to the large global memory latencies, such a ratio would make this implementation bandwidth limited within the GPU, and thus data reuse schemes need to be employed to increase performance. By making the assumption that beam coefficients do not need to change within small time frames, it is possible to reuse the same weights for all the time bins residing in a GPU buffer. This is especially true for wide beams, however does not hold when very narrow beams are required, or when tracking non-astrophysical objects such as satellites. This drastically reduces the number of coefficients which need to be generated and read from global memory within a thread block. We use this assumption to partition beam generation, where a 3D CUDA grid is mapped to the input space, with time bins varying in the x-dimension, frequency channel along the y-dimension and beam subset along the z-dimension. Thus each thread block generates a subset of the beams for a number of time bins, for a single frequency channel. Within each thread block, local beam accumulators are declared and stored in registers, one per beam, to which weighted antenna values are added. The antenna coefficients are loaded once per antenna group and stored in shared memory. This antenna partitioning is required as otherwise too much shared memory would be utilised, especially for large arrays, resulting in a decrease in occupancy and GPU compute resource utilisation.

Our kernel was originally implemented to match BEST-II backend specifications, most notably the output data format of the F-engine, which sends out antenna signals as 4-bit, two's complement, floating point, complex voltages. This format enables the packaging of each antenna value as a single byte, and groups of 4 antennas can in turn be packaged as 32-bit words. This is beneficial for the GPU kernel as each group can be loaded with a single memory request, which can be coalesced if they are accessed contiguously by a thread warp. For this reason, antennas are processed in groups of 4 in the beamformer, and complex coefficients are loaded in groups of the same size as well. In the inner-most loop, where these antennas are accumulated to form beams, this also has the effect of increasing instruction level parallelism, thus decreasing execution time. Breaking down antennas into groups of 4 has the additional benefit of making the kernel extensible to larger arrays without affecting performance. Therefore, we have effectively implemented a generic multi-beam beamformer which scales linearly with increasing bandwidth, number of antennas, number of frequency channels and number of

Algorithm 2 Coherent beamforming GPU implementation

Require: input, output, coefficients, nsamp, nchans, nants, nbeams

Declare *coeffs* shared memory

for time bins to process **do**

Declare local beam accelerators *beams*[*beams_per_tb*]

for *ant* = 1 **to** *nants*/4 **do**

Load antenna group from global memory

Cooperatively load required coefficients and store in *coeffs*

Synchronise threads

for *b* = 1 **to** *beams_per_tb* **do**

Update local beam accumulator

end for

end for

for *b* = 1 **to** *beams_per_tb* **do**

Store generated beam to global memory

end for

end for

output beams.

5.2.1 Performance and Benchmarks

Algorithm 2 provides a detailed breakdown of our implementation. The algorithmic design is similar in concept to our direct dedispersion kernel, with two configuration parameters determining the number of threads per blocks and the number of local accumulators where antenna values are combined to form beams. Performance benchmarks demonstrate that increasing both their values yields a performance benefit until a global maximum is reached, after which performance starts to degrade. Figure 5.2 determines the optimal values for these parameters on the test device, an NVIDIA GTX 670, where 16 accumulators and 128 threads per block provide the best performance. For these tests a 420 ms simulated buffer containing 32 4-bit complex sampled antenna voltage streams having a 20 MHz bandwidth channelised into 1024 frequency channels, was generated and processed 10 times for each combination. The resulting mean was used as a measure of

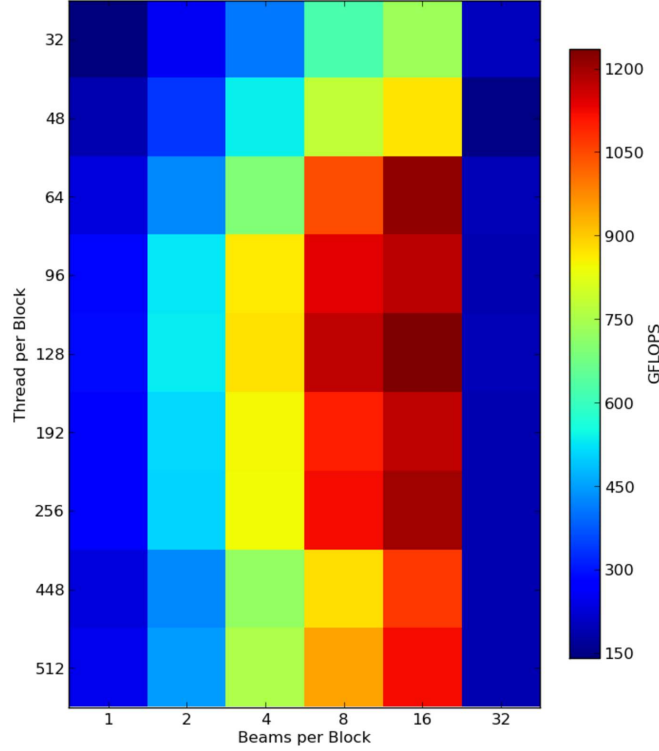


Figure 5.2: Configuration optimisation for the beamforming kernel, where 128 threads per block, each generating 16 beams, provides the best parameter combination on the test device.

the configuration efficiency. The optimal configuration is capable of sustaining more than 1.2 TFLOPs when 16 or more beams are generated, which amounts to approximately 50% of the device’s peak theoretical performance. This kernel’s limiting factor is the high number of shared memory requests needed to access phase weightings for each beam, antenna and channel triplet.

A scalability analysis of the algorithm was also performed, where a 420 ms data buffer containing voltage data from 32 single polarisation antennas with a bandwidth of 20 MHz, channelised into 1024 frequency channels, was used to generate an increasing number of synthesised beams in one GPU iteration. Figure 5.3 shows that performance scales linearly with increasing number of beams and the execution time is a fraction of realtime for BEST-II parameters. The maximum number of output beams which can be optimally synthesised is determined by the amount of global memory available on the GPU. The GPU spends 50% of its time waiting for raw voltage data to be copied from host to GPU memory, thus indicating that the implementation is bandwidth limited over the PCIe link. The kernel is not affected by the number of frequency channels for a given bandwidth.

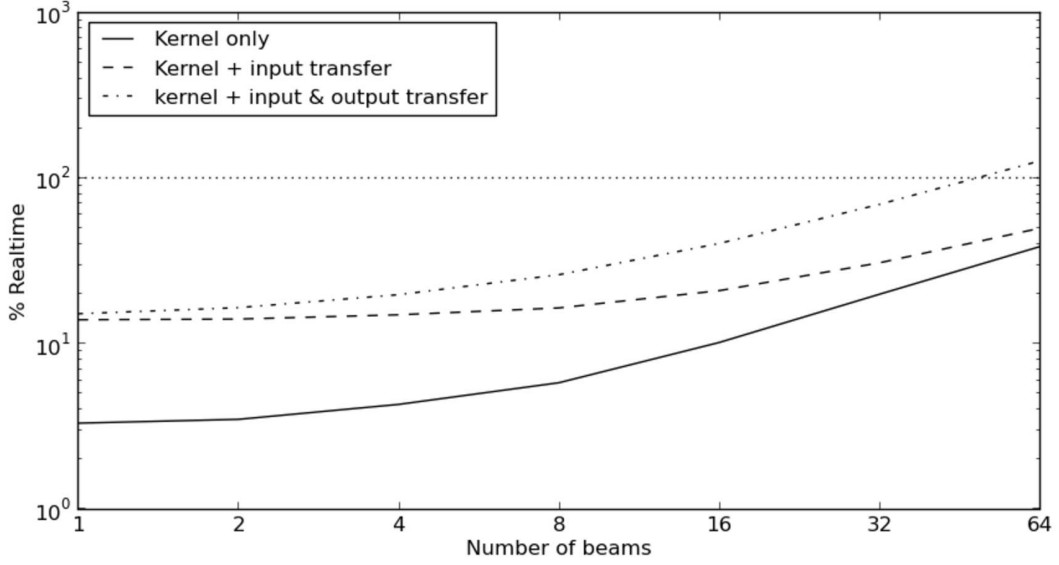


Figure 5.3: Coherent beamforming performance benchmarks for varying number of synthesized beams, for BEST-II parameters.

We have also benchmarked the scalability of the beamformer for increasing number of antennas, the results of which are shown in figure 5.4. Several 150 ms complex voltage buffers, each containing inputs from an increasing number of single polarization antennas having a bandwidth of 20 MHz, channelised into 1024 frequency channels, were generated and beamformed into 1 and 16 beams in one GPU iteration. Performance scales linearly with increasing number of antennas. 16 beams can be synthesised in realtime for 256 antennas when excluding data transfer time. The transfer-to-compute ratio is approximately 1 across the entire range.

The above two benchmarks clearly demonstrate the PCIe bandwidth bottleneck for beamforming kernels. When excluding the time required to copy the generated beams out of GPU memory, the total execution time is evenly split between kernel execution and transferring antenna voltages to GPU memory. This data movement overhead can essentially be masked by using two CUDA streams if the pipeline is capable of overlapping kernel execution and data transfer for the following iteration, however this would also require additional global memory, which is a scarce resource. The situation is worse when the generated beams are post-processed externally to the GPU on which they were generated, and the pipeline will essentially be dominated by PCIe transfer overheads. This poses several challenges for GPU-based beamforming pipelines and raises doubts on whether this is

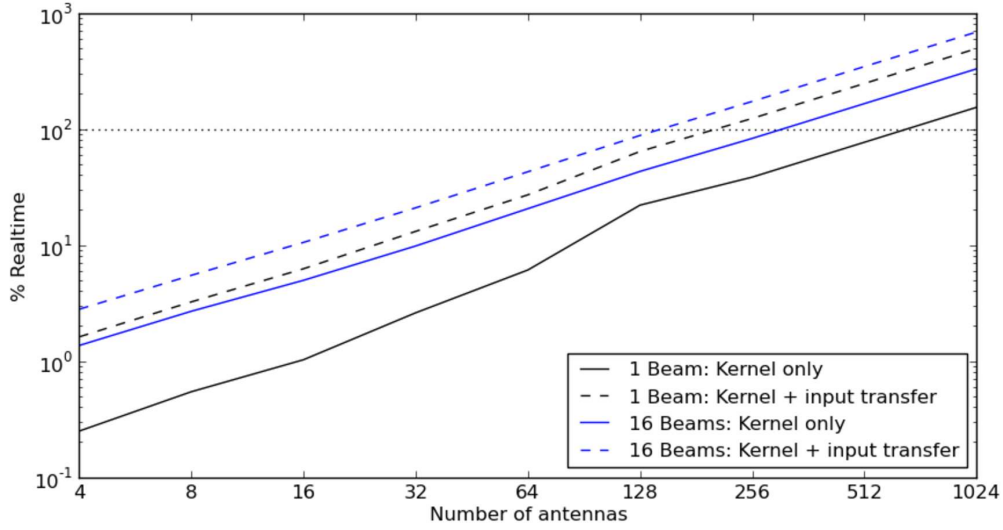


Figure 5.4: Coherent beamforming performance benchmarks for varying number of antennas when synthesising 1 (blue) or 16 (black) beams.

a viable solution for large- N aperture array telescopes. This problem can be somewhat alleviated by keeping the generated beams in GPU memory and performing further processing, thus increasing the compute-to-copy ratio. This can take the form of correlation, accumulation for generating images, or transient detection.

GPU-based beamforming is a relatively unexplored area in radio astronomy, possibly due to the assumption that any such system will be severely limited by the PCIe bandwidth required to transfer data to the GPU, as previously discussed. [Sclocco et al., 2012] have implemented a GPU-based version of their Blue Gene/P beamformer within the LOFAR pipeline, and state that they achieve a 45-50 time speed up, at 642 GFLOPs when using CUDA, on an NVIDIA GTX 580 (using 40% of the peak theoretical performance), with a power efficiency improvement of 2-8 times. In this section we have shown that our implementation achieves a higher GPU utilisation rate, being 25% more optimal than their quoted benchmarks. This utilisation rate is also sustained across a much wider parameter range (see [Sclocco et al., 2012, figure 12]). Nevertheless, both implementations suffer from bandwidth limitations. In chapter 6 we examine the scalability of this implementation to SKA₁low and SKA₁-mid.

5.3 Pipeline Integration

We have integrated our beamforming kernel into the transient detection pipeline described in chapter 3. The beamforming kernel has to be executed before the RFI excision and dedispersion stages, after which each beam is then processed by a separate GPU instance. The main challenge lies in determining the most efficient way to generate these beams, minimising as much as possible data movement between the host and GPUs, as well as amongst GPUs. Three basic schemes can be employed:

- The simplest scheme, implementation wise, would be to have each CPU processing thread generate its own coherent beam. This would require the input data buffer to be copied to each GPU instance, thus replicating this buffer multiple times within a GPU, greatly reducing the number of time spectra which would fit in global memory. As clearly indicated in figure 5.3, this would result in a severe degradation in performance due to the time spent transferring data over the PCIe links. Also, the beamforming kernel is optimised for generating multiple beams in parallel, and this scheme would not be fully utilising the kernel’s performance capabilities.
- At the other extreme, a single GPU can generate all the beams required by all the processing threads and then copy each beam to its destination, which could be on a separate GPU. Only one host to GPU transfer is required, and kernel execution time is minimised when compared to the scheme above where multiple kernel launches are required, each generating one beam, which is clearly inefficient. However, a copy per generated beam is required, some of which can be across GPUs for multi-GPU systems. This scheme also introduces heterogeneity across GPUs in the pipeline, where one GPU acts as a master which provides clients with data to work on.
- An alternative approach is to combine both schemes, where the beams required by all the processing threads allocated to the same GPU are generated in one kernel launch on the GPU itself. The input antenna voltages need to be copied once to every GPU, where a “master” thread launches the beamforming kernel, generating all the required beams. After completion each processing thread is provided with a pointer to its input beam, and processing advances as per standard transient detection.

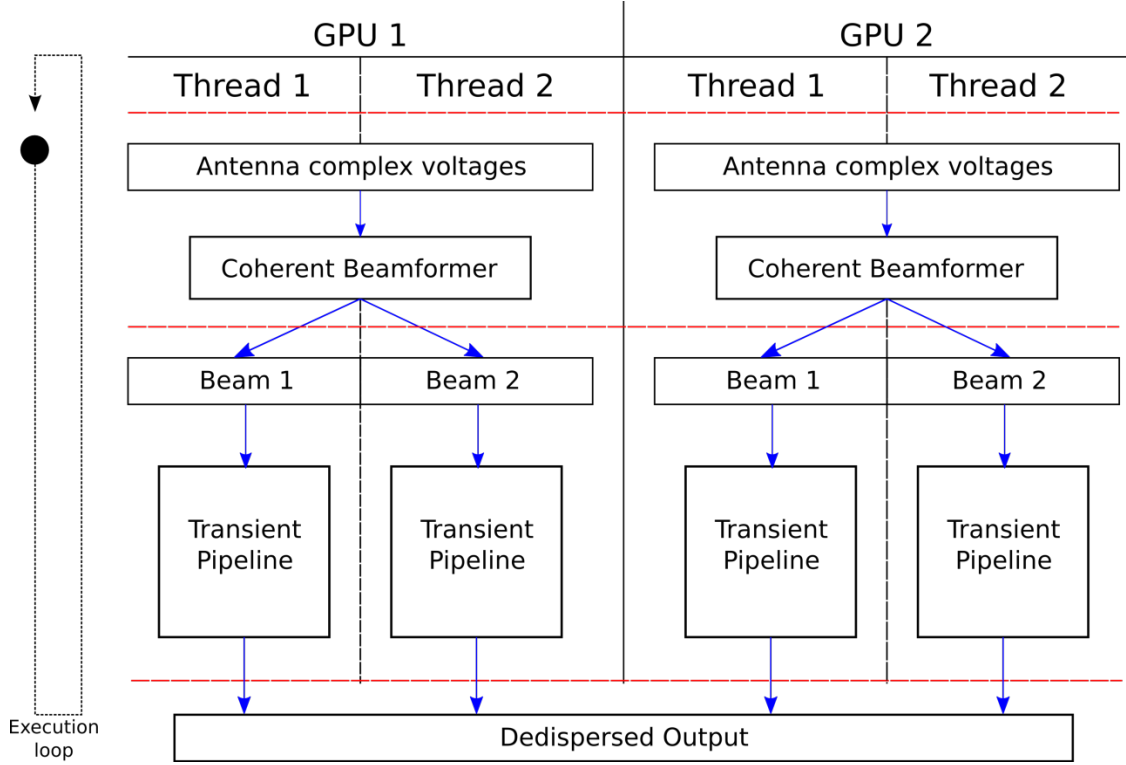


Figure 5.5: A schematic of the transient detection pipeline described in chapter 3 with integrated coherent beamforming kernel.

The last scheme above provides a good balance between input data replication, data transfers over PCIe links and beamforming kernel efficiency, and is the scheme we implemented for our pipeline. The resulting design is illustrated in figure 5.5. The buffered antenna voltages are copied to an input buffer allocated on every attached GPU. These transfers occur in parallel over different hardware PCIe links, thus maximising the available motherboard PCIe bandwidth with minimal overhead. A master thread per GPU (belonging to the first allocated beam) is responsible for generating all the GPU buffers and launching the beamforming kernel. The kernel will generate all the beams in a shared output buffer on the GPU, which can be accessed directly from other processing threads allocated on the same GPU, thus not requiring a copy per generated beam. The synthesised beams are then processed in place by independent CPU threads.

The main shortcoming of this design is that the “slave” processing threads have to wait for the beamforming kernel in the master thread to finish before they can start processing data, thus wasting valuable clock cycles. This duration is relatively short when compared to the time required for dedispersion. In the current implementation, the slave threads are blocked in a barrier synchroniser. An

alternative implementation would have these threads perform somethings useful in the meantime, such as phase/gain calibration. The complex coefficients are computed by the pipeline manager whilst waiting for the processing threads to finish working on the previous input buffer. This allows beam pointings to change dynamically during pipeline execution, which is useful for tracking observations or online follow-up of interesting transient events, provided appropriate feedback mechanism exist within the pipeline itself. The weight computation overhead is negligible relative to the execution time of a single pipeline iteration. The array configuration is stored in an XML file containing relative antenna locations.

5.4 Processing analysis of beamforming pipeline

In order to test the correctness of the beamforming kernel, and in order to benchmark the performance of the entire pipeline, we set up a mock BEST-II backend, where the pipeline was deployed on a GPU server connected to a ROACH running the BEST-II F-engine. Due to FPGA resource limitations, a SPEAD packetisation block could not be included in the F-engine design, so the XAUI output stream was simply branched to a 10GigE block, with UDP headers encapsulating groups of 128 time samples from 32 antennas, for each frequency channel, thus resulting in 4 KB packets with an additional 64-bit header containing the timestamp of the first time bin in the packet, as well as the frequency channel index. The total output data rate can be calculated using $D = C \times T \times A \times W$, where C is the number of frequency channels, T is the number of samples per second, A is the number of array elements and W is the word length. In our case, $C = 1024$, $T = 19531.25$, $A = 32$ and $W = 8$ bits (4-bits for each complex component), resulting in a total output bandwidth of 5.12 Gbps, excluding packet headers. Therefore a single 10GigE link is sufficient for data transfer between the F-engine and GPU server. At the receiving end, the packet receiver described in section 4.2.1 was updated to be compatible with this format. Groups of 128 time samples are considered as heaps, and thus heap functionality is still applicable. An additional lookup table is required to match the antenna order sent by the F-engine to the element ordering within the array and XML file, and is performed in the buffering thread.

A beam response test was conducted in order to test the correctness of the beamforming kernel. The 32 ADC inputs were left unconnected such that they only measure background and instrument noise. The signal from one of the inputs was mirrored across all connections, resulting in identical antenna signals. Signal

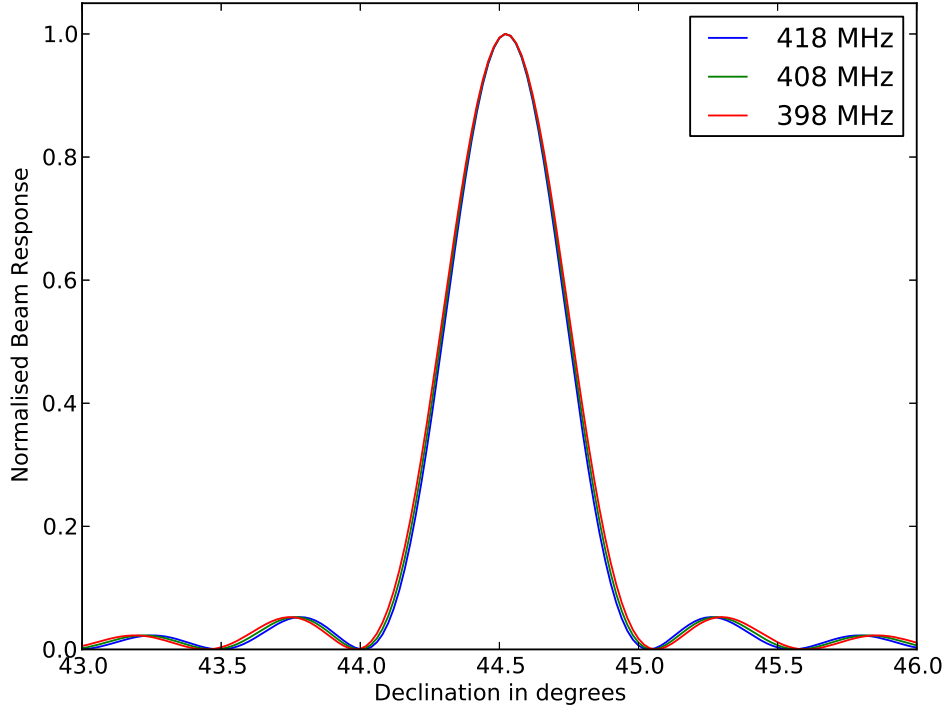


Figure 5.6: Beam pattern for the central and edge frequency channels for 6° around the zenith.

equilisation and gain calibration in the F-engine scales up this noise to use the entire bit width dynamic range. These signals were then phase calibrated to point to the zenith and transmitted continuously to the GPU server. The beamforming pipeline was set to generate 4 beams per iteration, each having a pointing difference of 0.02° , where the synthesised beams were written to disk for offline processing. BEST-II antenna positions were used to generate the beam coefficients, with the zenith at a declination of 44.524° . The current implementation can only generate pointings along the y-dimension, so for this test the interference pattern should resemble an 8-slit system with a central beam width of approximately 1° and 7 nulls between the central beam and first grating lobe. The resultant beam pattern, showing the central beams and two nulls, is shown in figure 5.6.

The entire pipeline was also benchmarked, using BEST-II parameters as a reference telescope, with observational and survey parameters based on those used to generate timings for table 3.3. A 2.52 s buffer containing voltage data from 32 20 MHz single polarisation antennas centred at 408 MHz, split into 1024 frequency channels, was generated and processed in a single pipeline iteration. One NVIDIA GTX 660 Ti card was used for this test. Four synthesised beams were generated

Antenna Voltage Copy Time*: 291.92 ms	
Beamforming*	74.98 ms
Bandpass Fitting	80.16 ms
RFI Thresholding	36.44 ms
Dedispersion	1780.6 ms
Median Filtering	70.96 ms
Detrending	44.16 ms
Copy from GPU: 101.64 ms	
Total iteration time: 2379.27 ms	

Table 5.1: GPU timings for one pipeline iteration performing beamforming, bandpass fitting, RFI thresholding, dedispersion, median filtering and detrending.

* Performed by the master thread on every GPU

from this buffer, each of which was then dedispersed over a range of 864 DM trials with a maximum DM of 86.4 pc cm^{-3} . Table 5.1 lists the accumulated execution time of each stage (since no kernel parallelisation is possible on the test device). The GPU to CPU transfer of antenna voltages and the beamforming kernel launch is performed by the “master” processing thread, and the input data buffer is limited by the amount of global memory available. This table shows that, for a 32 element array, the beamforming cost is negligible when compared to the total execution time of the entire pipeline, and amounts to approximately 3%. This hints to the possibility of deploying a fully GPU-based beamforming and transient detection system for small, low bandwidth arrays. Analogue signal reception, digitisation, channelisation and equilisation still need to be performed prior to beamforming, and these operations are more suitable to an FPGA-based system, such as the F-Engine deployed at the BEST-II array.

5.5 Conclusion

We have presented a GPU implementation of a coherent multi-beam beamformer which can synthesise a number of coherent beams for arrays with an arbitrary number of elements and frequency channels. The kernel utilises 50% of the peak theoretical performance on the test device, at 1.2 TFLOPs, and is limited by shared memory bandwidth, which is required to load complex beam coefficients. The overall implementation is limited by PCIe bandwidth, where for a serial execution pipeline, 50% of the processing time is spent transferring data from CPU to GPU memory. If the beams need to be post-processed outside of the GPU on which they

were generated, the GPU remains mostly idle, waiting for data transfers request to terminate.

The latter can be alleviated by performing additional operations after beam generation, and to test this assertion we have integrated this kernel with the transient detection pipeline described in chapter 3. A mock BEST-II backend running just the F-Engine was set up and used to benchmark this system, and we show that for small, low bandwidth arrays the computational cost for beamforming is negligible when compared to dedispersion. This gives rise to potential systems where these operations are integrated, allowing for dynamic observations where beam pointings can be updated online, in real time, triggered by interesting events during the event detection stage of the transient detection pipeline. Having the raw voltage data available in memory can potentially enable on-the-fly correlation and image generation for more accurate transient source localisation.

In the next chapter we determine the hardware resources required to scale up such a pipeline to larger arrays, specifically to single station, as well as central processing station, specifications for SKA₁-low and SKA₁-mid.

CHAPTER 6

APPLICABILITY OF GPUS TO SKA₁

The Square Kilometre Array (SKA) provides an excellent opportunity for searching for fast radio transients. The increased sensitivity, field of view and survey speed of the SKA, compared with other radio telescopes, will make it an ideal instrument to search for impulsive emission from high energy density events. The three telescopes making up the SKA will, however, pose considerable processing and data transport challenges. In this chapter we examine some computational aspects of the SKA, concentrating mostly on beamforming and dedispersion.

Recently, [Dewdney, 2013] revised the specifications for phase 1 of the SKA, and we use the configuration and parameters defined here as a basis for the analysis presented in this chapter. We start off by introducing the SKA, discuss possible advances in technology up till construction of the SKA, and describe methods for computing transient search parameters. We then analyse the applicability of GPUs for SKA₁-low station beamforming and channelisation, as well as present a possible use case, together with estimates for computational resources required, for SKA₁-low station-level transient searches. GPU-based implementation for several stages of the SKA₁-mid non-imaging pipeline are also investigated, focusing on beamforming, channelisation and dedispersion.

6.1 The Square Kilometre Array - Phase 1

The scientific motivations for building the SKA¹ are highlighted by Key Science Projects (KSPs), which represent unanswered questions in fundamental physics, astrophysics and astrobiology that the SKA will play a key role in addressing [Carilli and Rawlings, 2004, Gaensler et al., 2013]. Of particular significance to

¹ <http://www.skatelescope.org/>

this thesis is the potential use of the SKA for probing the fast transient phase space. The SKA will provide continuous frequency coverage from 50 MHz up to 10 GHz in the first two phases of its construction, with phase 1 providing $\sim 20\%$ of the total collecting area at low and mid frequencies, and phase 2 seeing the completion of the full array. The major observatory entities will be located in Australia (SKA₁-low and SKA₁-survey) and South Africa (SKA₁-mid), each hosting a supercomputing facility termed the Central Signal Processing (CSP), which will combine and process the signals from all the stations of their respective continent. The SKA₁ system baseline design [Dewdney, 2013] provides a baseline technical starting point and system decomposition for eventual construction of the SKA, and all listed technical specifications and calculations are based on values from this document, unless otherwise stated. We now proceed to give an overview of two of the telescopes comprising SKA₁. There are no plans to include a non-imaging infrastructure for SKA₁-survey, so it will not be included in the following discussions.

6.1.1 SKA₁-low

The main science goal of the low frequency SKA is to study the highly redshifted 21 cm HI line, and key design decisions have largely followed the recommendation of the SKA-EoR science working group. It will also be well suited for conducting low radio frequency observation of pulsars and, possibly, extrasolar planets. This will consist of an aperture array of $\sim 250,000$ log-periodic, dual-polarised antenna elements, most of which will be arranged in a very compact configuration (the *core*) with a diameter of ~ 1 km, the rest being arranged in 35 m diameter stations, configured as three equally spaced spiral arms, with a maximum radius of ~ 45 km. The antenna array will operate from 50 MHz to ~ 350 MHz, with peak sensitivity at 108 MHz, and a sensitivity of ~ 1000 m²/K at the zenith, assuming an instantaneous bandwidth of 250 MHz, and a core brightness temperature sensitivity of ~ 1 mK. The elements will be grouped into 911 35-m stations (866 in the core, 45 in the spiral arms). Table 6.1 lists the key specifications of SKA₁-low, as defined by [Dewdney, 2013]. The station beamformer will generate a single, smooth beam with a FoV of 20 deg². Signals from these beamformers will be transported to the CSP, where they will be channelised and cross-correlated. In this chapter we'll examine the applicability of GPUs for station-level beamforming, as well as examine the possibility of conducting station-level fast transient searching.

Lower Frequency	50 MHz
Upper Frequency	350 MHz
Bandwidth	300 MHz
Antennas per Station	289
Number of Stations	911
Total Antennas	263,279
Antenna Area	2.25 m ²
Station Diameter	35 m
Station filling factor	0.7
Total Physical Aperture	8.0×10^5 m ²
Channel Resolution	1 kHz
Maximum Number of channels	250,000
Number of Beams	1
Inst. BW per beam	250 MHz

Table 6.1: Key specifications of SKA₁-low, as defined by [Dewdney, 2013]

6.1.2 SKA₁-mid

This telescope will primarily address observations of the 21-cm hyperfine line of neutral hydrogen, many classes of radio transients, and a survey of the entire visible sky for pulsars with a pseudo-luminosity depth of 0.1 mJy kp² at 1400 MHz out to a distance of 10 kpc. For this survey, regions defined by dispersion measure will determine the optimum frequency ranges for the Galactic plane, the Galactic centre and high Galactic latitudes. The array will be a mix of 64 13.5 m diameter dishes from the MeerKAT² array and 190 15 m SKA₁ dishes, arranged in a moderately compact core with a diameter of ~ 1 km (to support pulsar searching) and a further 2D array of randomly placed dishes out to ~ 3 km radius. Three spiral arms will extend to a radius of ~ 100 km from the centre. SKA₁-mid will cover a frequency range of 350 MHz to at least 3050 MHz in three receiver bands, with lower frequency receivers having a bandwidth of ~ 1 GHz and higher frequency receivers ~ 2.5 GHz in each polarisation. SKA₁ dish sensitivity is expected at 6.9 m²/K, with a combined array SEFD of 1.7 Jy. Table 6.2 lists the key specifications of SKA₁-mid, as defined by [Dewdney, 2013]. Only SKA dishes were included for these calculations, and all processing requirement calculations in the rest of this chapter will focus on these as well. Also, only the lower three bands are listed, as these frequency regions will be used for transient-related science, except for Galactic centre observations which require higher frequencies. For transient

² <http://www.ska.ac.za/meerkat/index.php>

	Band 1	Band 2	Band 3
Frequency Range (MHz)	350 - 1050	950 - 1760	1650 - 3050
Max. Available BW (MHz)	700	808	1403
Aperture Efficiency	~60%	~65%	~78%
Average T_{sys} (K)	28	20	20
SEFD* (Jy)	4.4	2.1	2.1
Minimum detectable flux ($\mu\text{Jy s}^{-1/2}$)	105	58	54
$A_{\text{eff}}/T_{\text{sys}}$ (m^2/K)	779	1309	1309
FoV @ centre frequency (deg^2)	2.8	0.75	0.25
SSFoM* ($10^9 \text{ m}^4 \text{ K}^{-2} \text{ deg}^2 \text{ MHz}$)	1.2	1.0	0.08

Table 6.2: Key specifications of SKA₁-mid, including only SKA dishes, as defined by [Dewdney, 2013]

* For all antennas

searching, signals from the dishes will be transported to the CSP, where they will be beamformed and passed through specialised transient search equipment.

6.2 Technology Configuration

In section 2.2 we have discussed the current state-of-the-art in processor and accelerator technology, focusing on three devices: the Intel Xeon E5-267, NVIDIA K20 and Intel Xeon Phi SE10. Construction of SKA₁ will start in 2016, and a technology freeze date of 2016 is assumed for commercially available computer equipment which will be used for initial science data processing. Due to the dynamic nature of technology, it's difficult to predict the processing specifications of future devices. Here we'll list several assumptions which will be used in the rest of this chapter, with a primary focus on GPU technology.

GPU performance has roughly followed Moore's law in the past few years. Current GPUs can sustain a peak theoretical performance of ~ 3.5 TFLOPs with an internal memory bandwidth of ~ 280 GB/s. We therefore project that by 2016, GPUs will have a peak theoretical performance of ~ 10 TFLOPs, retaining a GPU dissipation of ~ 250 W. The new upcoming NVIDIA architecture, Maxwell, will introduce Unified Virtual Memory, which will allow the CPUs and GPUs on a host system to see all of system memory, simplifying data transfer between the two. The following architecture, Volta, will provide an internal memory bandwidth of 1 TB/s, greatly improving the performance of bandwidth limited implementations. We will also assume that PCIe 3 will be the interfacing platform for these de-

	% Peak Performance	Arithmetic Intensity	Bandwidth
Dedispersion	30%	3.6 TFLOPs	13.4 GB/s
Beamforming	60%	7.2 TFLOPs	13.4 GB/s

Table 6.3: Scalability measure for direct dedispersion and coherent beamforming on NVIDIA Volta devices.

vices, which supports 15.75 GB/s bi-directional data rate over 16 lanes, as the specifications for PCIe 4 are not due until 2015.

We will also assume that the ratio between CPU and GPU power remains relatively constant, with a proportional increase in compute power. Host memory will probably be composed of DDR4 SDRAM, which will feature faster clock frequencies, lower voltages and 230 Gbps of maximum bandwidth for a 72-bit wide data bus. No doubt, this will greatly decrease the latency for internal memory transfers, and prove beneficial when transferring high throughput data from network adapters. These will probably consist of 40 Gbps Ethernet or 56 Gbps Infiniband adapters. We follow [Dewdney, 2013] and choose the latter, with an additional benefit being that dual-port 56 Gbps adapters will match the maximum PCIe 3.0 transfer rate.

We now apply the specifications listed above to the algorithm implementations discussed in the previous chapters, particularly to direct dedispersion and coherent beamforming, to create a “scalability measure” for each. The current implementation of the dedispersion kernel can achieve $\sim 25\%$ of the peak theoretical performance on Kepler devices, as opposed to the 10% listed in [Dewdney, 2013]. On Volta devices, this will get an additional boost due to the increased memory bandwidth, which will probably lead to a re-design of the algorithmic behaviour so as to reduce the reliance on shared memory and take advantage of increased global memory bandwidth. We’ll therefore assume that $\sim 30\%$ of Volta GPUs’ peak theoretical performance can be utilised, and a 16-lane PCIe 3 interface for transfer between the CPU and GPU. The beamforming kernel achieves $\sim 50\%$ of the peak theoretical performance, and the kernel will probably gain a further boost due to the increased internal memory bandwidth, so we’ll assume that 60% of Volta GPUs’ peak performance can be utilised. These scalability measures are listed in table 6.3. Additionally, empirical testing on current GPUs show that GPU PCIe bandwidth utilisation is approximately 90% of specified peak.

6.3 Transient Search Parameters

In the following sections we'll be discussing the computational and hardware requirement for conducting transient surveys with SKA₁. Here we develop a general framework for calculating the surveying parameters which will be used throughout this chapter. Blind surveys of large total solid angles require full-FOV sampling, which is feasible only for a subarray comprising the innermost antennas, the core array. This is characterised as a circular distribution of n_a antennas with diameter b_c . The core array contains a fraction $f_c \equiv n_a/N_a$ of the total number of antennas. The maximum number of independently pointed, coherently combined beams, N_b , which can be formed within the FoV of a single element beam is given by:

$$N_b = \frac{\Omega_0}{\Omega_{\text{arr}}} = \left(\frac{D_{\text{arr}}}{D_0} \right)^2 \quad (6.1)$$

where Ω_0 is the single element FoV, Ω_{arr} is the array beam FoV, D_{arr} is the diameter of the entire array, D_0 is the element diameter.

In this chapter we will only consider the computational requirements for coherent signal combination. Incoherent signal combination and subarraying will yield different event detection rates and have different processing requirements, as discussed by [Colegate and Clarke, 2011]. Irrespective of the signal combination mode used, each synthesised beams needs to be searched independently for transient events. Here we will focus on post-detection analysis for fast transient, single-pulse blind surveys. Dispersion and scattering effects can be used to define the bounds of the search.

The maximum DM value, DM_{max} , and consequently the distance to which the search can be conducted, is limited by

- the degree of expected scatter broadening, which can be computed by using the empirical relationship defined in equation 1.17
- The amount of signal smearing within one frequency channel, where restricting this to one time sample results in a DM_{max} of

$$\text{DM}_{\text{max}} = \frac{\Delta t \cdot f_0^3}{8300 \cdot \Delta f} \quad (6.2)$$

where Δt is the sampling time, f_0 is the frequency at the lower edge of the band and Δf is the channel width. This is a rearranged version of equation 1.15.

A model of the galactic distribution of free electrons, such as the NE2001 model ([Cordes and Lazio, 2002]) can also be used to restrain this parameter, and is the scheme adopted by [Dewdney, 2013]. For extragalactic surveys, the updated fit by [Lorimer et al., 2013] will be used. Higher DM values can be searched for by altering the beamformed time series, such as downsampling by combining adjacent time samples.

The required spectral resolution depends on the amount of DM smearing within a single channel for DM_{\max} . By setting a limiting factor on this smearing amount, F_{smear} , the total number of frequency channels can be computed. Unless explicitly stated, we will not include the computational requirements for channelisation in our calculations, however this scales as $\mathcal{O}(N_t N_p (\log_2(N_c) + N_{\text{taps}}))$ whereas dedispersion scales as $\mathcal{O}(N_t N_c N_p N_{\text{DM}})$ (when direct dedispersion is used), therefore the total computational requirements are dominated by dedispersion. Here, N_t is the number of time samples, N_p is the number of polarisations, N_c is the number of channels, N_{DM} is the number of dispersion measure trials and N_{taps} represents the number of taps on a presumed polyphase filterbank channelisation implementation. N_c is therefore

$$N_c = \frac{N_{\text{sub}} \times \tau_c \times DM_{\max}}{F_{\text{smear}} \times W} \quad (6.3)$$

where τ_c is the pulse smear within a single channel prior to finer channelisation and W is the expected pulse width.

The DM step, ΔDM , depends on the expected signal width, and can be restricted by limiting the loss in S/N of a pulse due to dedispersion at an incorrect DM value. The interval should not be large such that a transient with a true DM lying between two trial DMs is significantly broadened, however a value which is too small will greatly increase the required computational resources. This value can be calculated by selecting the maximum ΔDM at which equation 3.5 yields a S/N loss less than a limiting factor DM_{smear} , where W_{ms} is replaced with the minimum expected intrinsic signal width. The number of DM trials is then simply the DM range divided by ΔDM .

Further restraints can be applied by relying on the fact that the amount of smearing due to scattering increases with DM. As soon as the scattering timescale of a pulse of width Δt becomes $2\Delta t$, the time series can be downsampled by a factor of 2, as the original high temporal resolution is no longer required. This process can be repeated until DM_{\max} is reached. This is the process which PRESTO uses

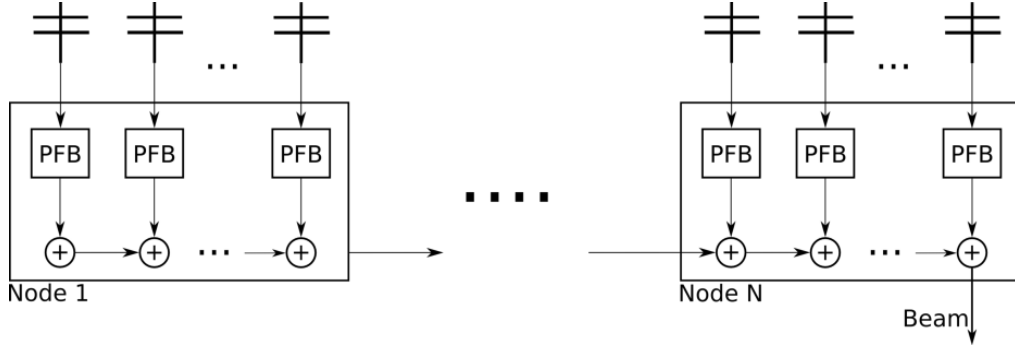


Figure 6.1: Ring-based beamforming schematic

to generate dedispersion plans, and has the effect of reducing the computational requirement with minimal effect on the sensitivity of the survey. For the purpose of this chapter, no downsampling will be applied to the time series, unless otherwise stated.

6.4 SKA₁-low station beamforming

Signals from each antenna within an SKA₁-low station need to be combined to generate station beams. Here's we'll concentrate on the applicability of GPU-based systems for station-level beamforming, limiting ourselves to the requirements defined in [Dewdney, 2013], who state that one output beam over the entire band is required per station. We'll investigate a setup where GPUs, together with their host systems, will perform all the required processing, including digitisation, channelisation and beamforming. We follow [Hickish, 2013] and assume a ring-based beamforming architecture, which essentially is a serial chain of addition stages whereby each processing node contributes to the formation of the entire beam. This setup is used by LOFAR [van Haarlem et al., 2012] to generate station beams. Figure 6.1 provides a visualisation of this scheme. We also assume Nyquist sampling with a 1.25 coding factor, resulting in 625 MSA/s. Assuming digitisation of antenna signals with n_s bit samplers, the input data rate to each processing node is

$$S_{\text{in}} = 2B\eta N_a N_p n_s \quad (6.4)$$

where B is the bandwidth, N_a is the number of antennas N_p is the number of polarisations and η is the coding factor. The output data rate is defined by the

Bits per Value	n_s	8
Coding Factor	η	1.25
Input Data Rate	S_{in}	~ 2.89 Tb/s
Output Data Rate	S_{out}	~ 10 Gb/s
Beamforming Operations	S_{ops}	~ 2.89 TFLOPs
Channelisation Operations	C_{ops}	~ 26.7 TFLOPs

Table 6.4: SKA₁-low station beamforming parameters

number of beams required, and is

$$S_{\text{out}} = 2B\eta N_b N_p n_b \quad (6.5)$$

where N_b is the number of beams and n_b is the bitwidth representing each beam sample. Since $N_b = 1$, S_{out} is significantly lower than S_{in} and therefore I/O bandwidth will be entirely dominated by the input data rate. The number of operations required for beamforming and channelisation are

$$B_{\text{ops}} = 8BN_b N_a N_p \quad (6.6)$$

$$C_{\text{ops}} = BN_b N_a N_p \cdot (\log_2(N_c) + 8N_{\text{taps}}) \quad (6.7)$$

where N_c is the number of frequency channels and N_{taps} represents the number of taps in a presumed polyphase filterbank channelisation implementation. The factor of 8 for N_{taps} represents the number of operations required for multiplying complex coefficients. A PFB implementation consists of two processing stages: complex multiplication with a pre-calculated function, and a FFT. The first stage can generally be optimally implemented, so we assign a 50% compute efficiency to it, whilst 30% is assigned to the FFT, based on performance benchmarks of the cuFFT library for a 1024-point single precision FFT. For the following calculations, we'll assume a 1024-channel PFB with 8 taps. Table 6.4 lists the values of the above parameters for SKA₁-low stations. Note that for ring-based beamforming, the input data rate to each node includes the output beamformed data from the previous node in the chain, and therefore the total node input data rate is $S_{\text{in}} + S_{\text{out}}$, where the latter term is constant for any number of input antennas.

Using the hardware specifications listed in table 6.3, figure 6.2 shows how many antenna signals can be processed by a single GPU (assuming signal digitisation is performed on the host CPU). These plots take into consideration implementation efficiency for each algorithm as well data transfer efficiency over the PCIe

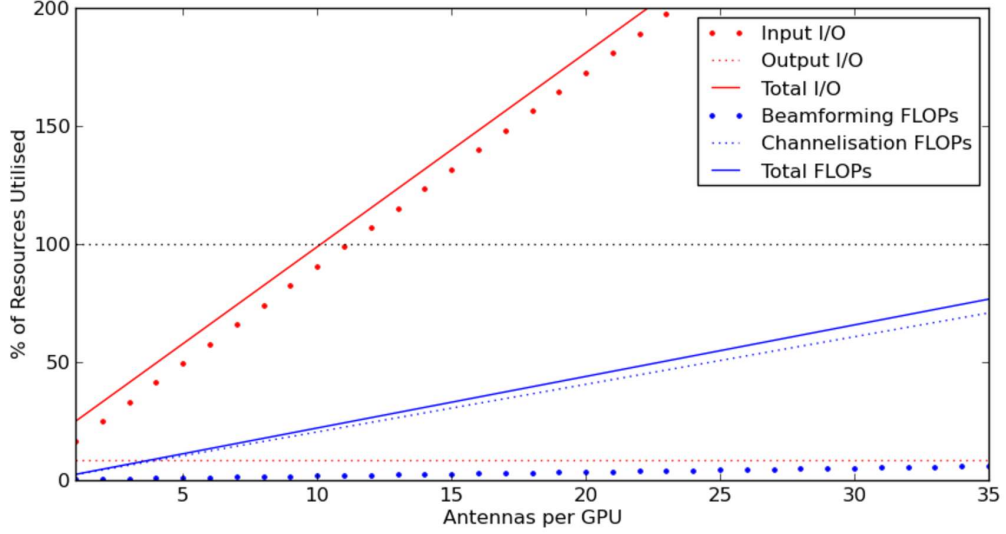


Figure 6.2: Resource utilisation for SKA₁-low station GPU-based processing

link. It is clearly evident that this setup is entirely limited by the data transfer rate onto GPU memory, with the GPU SP cores sitting mostly idle, even if coarse channelisation is performed at the station. Approximately 10 dual-polarisation antennas can be processed by a single GPU, thus requiring at least 29 GPUs per station, or almost 27,000 for all SKA₁-low stations. This should be contrasted by the number of Virtex 7 FPGAs which would be required, which would be 12 per station, or $\sim 11,000$ in total (see [Hickish, 2013]). With the latter option the computational resources on the chip are almost fully utilised.

We now present an approximate cost analysis for this setup, using pricing calculations provided by [Zarb-Adami et al., 2013]. Table 6.5 summarises the cost per station, as well as an aggregate cost for all the SKA₁-low stations, including deployment and 10-year operational costs. These figures exclude communication fabric and hardware, racking, cabling, labour and additional infrastructural costs. Two GPUs per server are assumed, since a dual-port 56 Gbps network adapter is required per GPU in order to fully saturate the PCIe links, requiring a total of 4 PCIe x16 slots. Clearly, this is a costly setup, with very high running costs, primarily due to high GPU power consumption. FPGA-based solutions might result in comparable deployment and replacement costs, however they provide a much higher performance per Watt, with power consumption almost one order of magnitude lower than GPUs. A more detailed cost-benefit analysis should be conducted in order to determine the best approach for station beamforming and channelisation.

Server Setup	2 GPUs + Host CPU
GPU Cost	1500 per unit
Host Cost	700
Cost per Server	3700
GPU Power Consumption	250 W
Host Power Consumption	250 W
Power Cost (Euro/Watt/Year)	2.5
Replacement parts	5% / year
Station Deployment Cost	55,500
Station Replacement Cost	562,500
Station Power Cost	150,000
SKA ₁ -low Deployment Cost	50,560,500
SKA ₁ -low Replacement Cost	25,280,250
SKA ₁ -low Power Cost	136,650,000

Table 6.5: SKA₁-low station processing costs, based approximately on numbers given by [Zarb-Adami et al., 2013]. Power and replacement costs are for a 10-year operational lifespan. All costs are in 2013 Euro.

6.5 SKA₁-low Station-level transient searching

In this section we examine the possibility of conducting station-level fast transient surveys in “piggy-back mode”, where the complex voltages from all the antennas are first coarsely channelised by an FPGA or GPU-based processing backend and sent to a network switch, where GPU-based processing nodes then combine the antenna signals and run the transient detection pipeline. This would be very similar to the ARTEMIS use case for international LOFAR stations, where each station becomes an independent surveying instrument. The digital backend will also form the station beam as required by the central SKA₁-low observation, whilst the beam generated by the GPU system can be pointed anywhere within the antennas’ FoV, allowing separate manoeuvrability.

In an ideal scenario, data is transferred only once to GPU memory, as additional transfers would increase the number of GPUs required. However, simply partitioning the set of antennas into isolated clusters would drastically reduce the sensitivity of the instrument. A better alternative would be to split the observing band, where each GPU receives signals from all antennas for specific subbands. This would reduce sensitivity by $\sqrt{(N_{\text{subs}} - 1)N'_c}$, where N'_c is the number of channels per subband, which can be countered somewhat by lowering the detection threshold during post-processing, as discussed in section 3.6. In the previous sec-

tion we stated that 29 GPUs would be required to process all the raw antenna signals. By assuming that after signal digitisation, channelisation and equalisation the total output data rate becomes $BN_aN_p n_s$, the number of required GPUs is then reduced to 12. Rounding this value up to the nearest power of 2, 16, provides us with the minimum channelisation requirements for splitting up the band.

For this section we'll assume the observing band is split into 16 frequency subbands, the same number as the minimum number of GPUs required to keep up with the input data rate, however the process below can be repeated for any number of subbands, as will be required by the final SKA₁-low design. The number of DM trials which can be processed, when using direct dedispersion, can be computed by calculating the “left over” computational resources after GPU beamforming and finer channelisation

$$\mathcal{F}_{\text{ops}} = \mathcal{F}_b + \mathcal{F}_c + \mathcal{F}_d \quad (6.8)$$

$$\mathcal{F}_b = 8e_bBN_aN_pN_b \quad (6.9)$$

$$\mathcal{F}_c = 8e_cBN_bN_p(\log_s(N_c) + N_{\text{taps}}) \quad (6.10)$$

$$\mathcal{F}_d = 4e_dBN_bN_{\text{DM}} \quad (6.11)$$

$$N_{\text{DM}} = \frac{\mathcal{F}_{\text{ops}} - \mathcal{F}_b - \mathcal{F}_c}{e_dBN_b} \quad (6.12)$$

where \mathcal{F}_b , \mathcal{F}_c and \mathcal{F}_d are the beamforming, channelisation and dedispersion operation counts in FLOPs/s respectively, and e_b , e_c , e_d are the beamforming, channelisation and dedispersion efficiency factors respectively. For SKA₁-low stations, $B = 15.625$ MHz (250 Mhz split across 16 subbands), $N_b = 1$, $N_p = 2$ and $N_a = 289$. The cost for summing across both polarisations prior to dedispersion is assumed to be negligible. The total channelisation cost is minimal due to the beamforming step, where all N_a antennas signals are collapsed into 1 beam. This would result in N_{DM} having a value of order $\mathcal{O}(10^5)$, suggesting that this scheme would either be memory or PCIe-bandwidth limited on the GPU.

The maximum buffering time, such that all of GPU memory is filled up in one transfer, with enough additional memory to store temporary output, is:

$$\tau_b = \frac{8 \cdot M_{\text{tot}} \cdot \alpha}{BN_p(n_sN_a + n_dN_b)} \quad (6.13)$$

where M_{tot} is the amount of GPU memory, α represents the percentage of available GPU memory (other buffers might be required by additional processing stages, or to be used as temporary buffers), n_d is the internal bit-representation of the

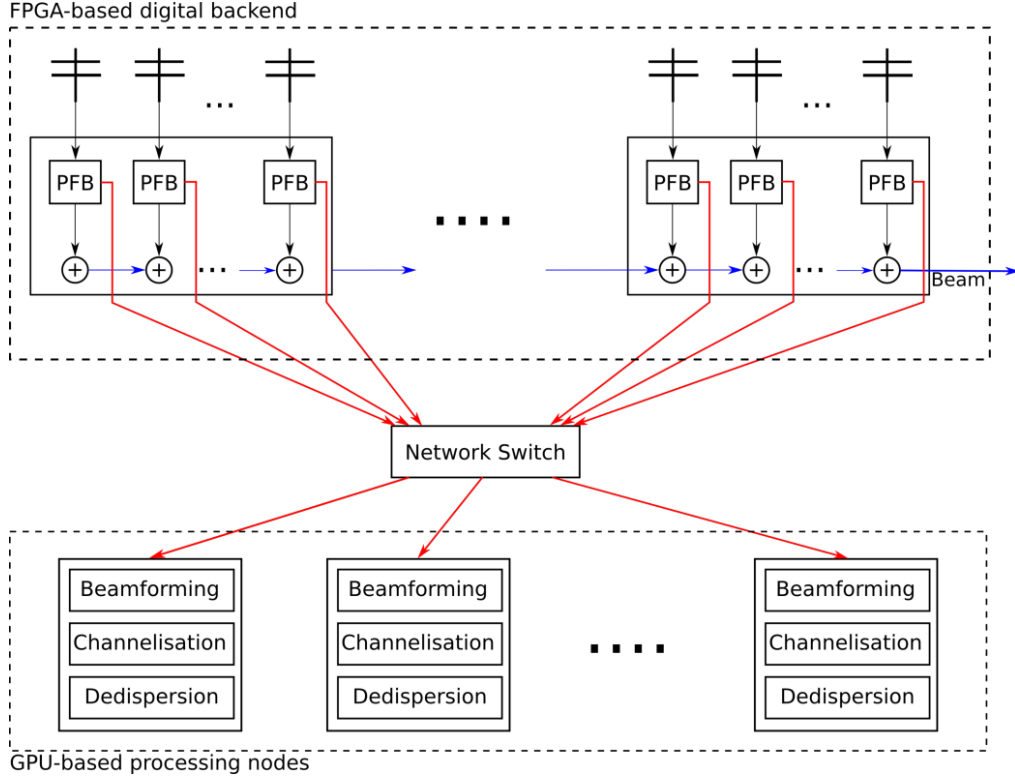


Figure 6.3: Ring-based beamforming schematic with transient detection pipeline. Red arrows represent coarsely-channelised data movement, whilst blue arrows

beamformed time series and τ_b is the buffering time. This would amount to a total input buffer size of

$$M_{\text{in}} = BN_a N_p \tau_b \quad (6.14)$$

The input buffer can then be copied to GPU memory in

$$\tau_i = \frac{e_t M_{\text{in}}}{B_{\text{PCIe}}} \quad (6.15)$$

where e_t is the PCIe bandwidth efficiency factor and B_{PCIe} is the available PCIe bandwidth. Using the relationships above, the number of DM values which can be processed in real-time can be calculated. The DM trials which can be processed in parallel is limited by the amount of available GPU memory, and this is equal to

$$N_{\text{DM-mem}} = \frac{N_c M_{\text{in}}}{B_0 n_d} \quad (6.16)$$

where B_0 is the number of samples for buffering time τ_b ($B = \tau_0 B$). The main assumption here is that the output dedispersed time series will be stored in the memory buffer used to store the input antenna voltages. Equation 6.3 can be used

to calculate a value for N_c , which primarily depends on the maximum DM value. The total number of processable DM trials within the available time frame is:

$$N_{\text{DM-proc}} = \frac{\tau_b - (\tau_i + \mathcal{T}_o \mathcal{F}_b + \mathcal{T}_o \mathcal{F}_c)}{4e_d \mathcal{T}_o B_0 N_b + \tau_d} \quad (6.17)$$

where \mathcal{T}_o is the inverse of the available GPU floating point resources for execution time τ_o and τ_d is the transfer time for a single dedispersed time series, equal to

$$\tau_d = \frac{e_t B_0 n_d}{B_{\text{PCIe}} N_c}. \quad (6.18)$$

The total number of DM trials processable in real-time is then

$$N_{\text{DM-tot}} = \min(N_{\text{DM-mem}}, N_{\text{DM-proc}}) \quad (6.19)$$

The above relationship will be dominated by τ_i , where assuming $\alpha = 0.95$ and $n_d = 8$ -bits, for SKA₁ station parameters, $\tau_b = 0.675$ s and $\tau_i = 0.306$ s. Based on these assumptions, a parameter table for each frequency subband was generated, a subset of which is shown in table 6.6. The full table is reproduced in appendix A, where each subband is processed by a single GPU. Treating each station as an independent entity would yield a very small DM range, especially at lower frequencies. However, these would generally operate in unison, pointed to the same coordinates in the sky. This allows the distribution of the DM range over the entire SKA₁-low telescope, with each station processing a subset of the full DM range. Alternatively, a more efficient dedispersion algorithm can be used, however we will concentrate on the direct dedispersion as a use case for station dedispersion.

The DM step and maximum DM value are computed as per section 6.3, assuming a minimum expected pulse width of 1 ms, 200% accepted scattering (such that a 1 ms pulse is scattered across 2 ms) and 150% pulse smearing. The DM fraction which can be processed on a single GPU is simply

$$\text{DM}_{\text{frac}} = \frac{\Delta \text{DM} \cdot N_{\text{DM}}}{\text{DM}_{\text{max}}} \quad (6.20)$$

Additionally, since the synthesised beam is generated on the GPU itself, it can be pointed anywhere within the antenna primary beam, thus enabling a wider surveying sky area. The last row in table 6.6 lists how many stations would be required to survey the entire station FoV, for the entire DM range, with the number

Low Frequency (MHz)	50	112.5	159.375	284.375
High Frequency (MHz)	65.625	128.125	175	300
Maximum DM (pc cm ⁻³)	510.185	887.71	1154.645	1699.194
DM Step (pc cm ⁻³)	0.000125	0.000915	0.004095	0.02328
Processable DMs	12,016	12,016	12,016	11,950
Number of Channels	15625	15625	15,625	8,192
Processable Max (pc cm ⁻³)	1.5023	10.996	36.114	278.186
DM Range Fraction (%)	0.003	0.013	0.043	0.164
#Stations Required	340	53	48	42

Table 6.6: Parameter subsets for SKA₁-low station blind, fast transient survey. Full table in appendix A.

of required beams changing per frequency subband. Following [Dewdney, 2013], the beam FoV, Ω_b , is

$$\Omega_b = \frac{\pi}{4} \left(\frac{1.3\lambda}{D_{\text{station}}} \right)^2 \quad (6.21)$$

where $D_{\text{station}} = 35$ m is the station diameter. The number of beams required is then simply $\lceil \Omega_a / \Omega_b \rceil$, where $\Omega_a = 20$ deg² is the antenna FoV. A subset of the core stations is enough for a full fast transient survey over the entire station FoV, for all the frequency subbands.

Using the values from table 6.5, figure 6.4 shows the cumulative deployment and running cost across frequency subbands, from higher to lower frequencies, for surveying the entire antenna FoV. These figures are server based, resulting in some stations processing a subset of the subbands. Clearly, this is a costly setup, and should be compared with the hardware and running requirements for performing the same processing at the CSP.

6.6 SKA₁-mid CSP Beamforming

Pulsar searching is one of the main science cases for SKA₁-mid, and various studies have been conducted on the search parameters and hardware requirements for this telescope (for examples, [Smits et al., 2008, Cordes, 2009]). [Dewdney, 2013] define the optimal configuration for pulsar searches, a summary of which is listed in table 6.7, where the worst-case scenario in terms of computational requirements is selected. In this section we focus on the beamforming requirements, whilst in the next we turn to dedispersion.

The dish output streams need to be channelised and beamformed prior to

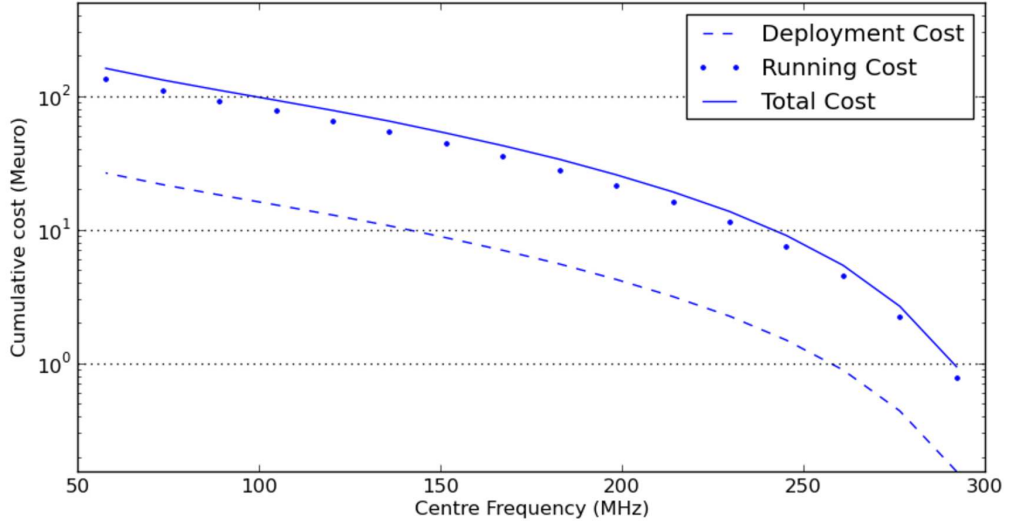


Figure 6.4: SKA₁-low station processing costs, based approximately on numbers given by [Zarb-Adami et al., 2013], showing cumulative cost across frequency sub-bands, from higher to lower frequencies. These are server-based, and the server number per station changes across the observing band.

passing through the transient detection pipeline. Since $N_a < N_b$, it is more efficient to channelise the observing band prior to beamforming, so channelisation can be easily integrated within the beamforming architecture. The input data rate to the SKA₁-mid beamformer, B_{in} , assuming 8-bits per complex value, is approximately 1.4 Tb/s, with an operation count, B_{ops} , of ~ 1.5 PFLOP/s and output data rate, B_{out} , of ~ 10.67 Tb/s, assuming a channelisation output sampling rate of $1/B$. The channelisation operation count, C_{ops} , is a small fraction of B_{ops} , amounting to about 15 TFLOP/s, or 26 GFLOP/s per dish, assuming an 8-tap polyphase filterbank channeliser and 20 kHz spectral resolution. Integrating the channelisation operations into the beamformer induces a minimal cost, even if dishes are channelised multiple times, depending on the beamforming architecture adopted. Simply matching these aggregate values to GPU specifications, taking into account the PCIe transfer and beamforming efficiency, e_t and e_b respectively, at least 350 GPUs would be required. However this is a highly conservative number, as it assumes the antenna signals are transferred once to a single GPU, which is an unlikely scenario.

We now provide a more realistic method for determining the number of GPUs required, assuming a series of ring-based beamforming chains, a schematic of which is depicted in figure 6.5. Dish signals are streamed to multiple nodes, each of which synthesise a subset of the beams. Node groups are chained together to form the

Telescope Parameters	
Optimal number of antennas	141
Associated array diameter	950 m
Centre frequency	1400 MHz
Bandwidth	300 MHz
Beam size at 950 MHz	$1.5 \times 10^{-4} \text{ deg}^2$
Number of beams	2222
Search Parameters	
Maximum DM	3000 pc cm ⁻³
Achievable maximum DM	827 pc cm ⁻³
Number of channels	15000
Sampling time	50 μ s
Frequency resolution	20 kHz
DM step	0.054 pc cm ⁻³
Number of DM trials	15350
Number of subbands	64

Table 6.7: Key specifications of SKA₁-mid, as defined by [Dewdney, 2013], representing the worst case scenario in term of computational requirements, corresponding to searches at galactic latitude $|b| < 5$.

beamforming ring. The beamformer output data rate to the rest of the pipeline can be reduced by computing the power of the voltage series and summing across polarisations directly after beamforming on the last GPU in each chain, resulting in an aggregate output data rate of ~ 5.34 Tb/s. Assuming no memory constraints ($\tau_b = 1$) and no execution overlap, the computational and I/O load of each node is

$$\tau_b = \tau_i + \mathcal{F}_c + \mathcal{F}_b + \mathcal{F}_p + \tau_o \quad (6.22)$$

where $\mathcal{F}_p = 4BN_p$ is the operation count for magnitude calculation and summing across polarisations, which is only performed by the last GPU in every beamforming chain, prior to transferring the beamformed series to the rest of the processing pipeline. It should be noted that input, processing and output operations can be overlapped on GPUs with dual copy engines, however since τ_b is dominated by τ_i the benefit will not be large, especially when considering the larger memory requirements and higher implementation complexity. $\tau_o = BN_b$ is the output data rate. τ_i includes dish inputs as well as the output from the previous node in the chain

$$\tau_i = 2BN_p(M_a + M_b) \quad (6.23)$$

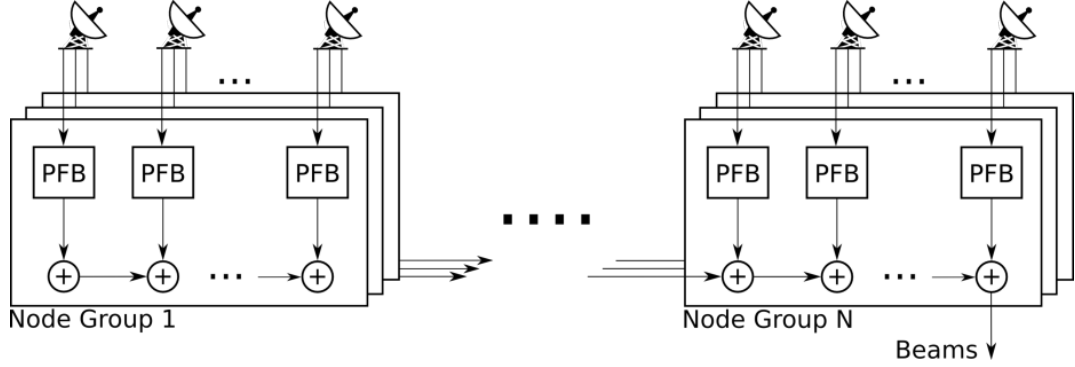


Figure 6.5: SKA₁-mid CSP ring-based beamforming schematic. Each antenna signal is streamed to multiple node groups, where each node within the group partially synthesise a subset of the output beams.

where $M_a \subseteq N_a$ and $M_b \subseteq N_b$. Ignoring algorithmic efficiency, arithmetic timing and I/O transfer efficiency, equation 6.22 becomes

$$2BN_p(M_a n_d + 4N_a(\log_2(N_c) + N_{\text{taps}}) + 4M_b M_a + M_b n_b) \leq 1 \quad (6.24)$$

Appropriate values for M_a and M_b need to be selected in order to minimise the difference between the sides of this equation, such as to maximise GPU resource utilisation. The total number of GPUs is then simply

$$N_{\text{GPUs}} = \left\lceil \frac{N_a \cdot N_b}{M_a \cdot M_b} \right\rceil \quad (6.25)$$

Figure 6.6 shows runtime values for various combinations, where the unit contour line represents the set of values for which GPU utilisation is optimised. Appropriate efficiency factors were included in these calculation, and a value for n_b of 8-bits was assumed. The colour mapping in this plot represents values for N_{GPUs} where the GPU utilisation rate is within 1% of the peak. The optimal values for M_a and M_b are 4 and 14 respectively, amounting to a GPU utilisation rate of 99.5%, 86.7% of which is spent transferring data over the PCIe bus. This is equivalent to about 5700 GPUs. By setting n_b to 4-bits, N_{GPUs} can be reduced to about 2700 ($M_a = 5$ and $M_b = 23$, with similar utilisation rates).

6.7 SKA₁-mid CSP Dedispersion

Each synthesised beam has to be independently dedispersed and searched for any interesting signals. Based on the search parameters listed in table 6.7, a total of

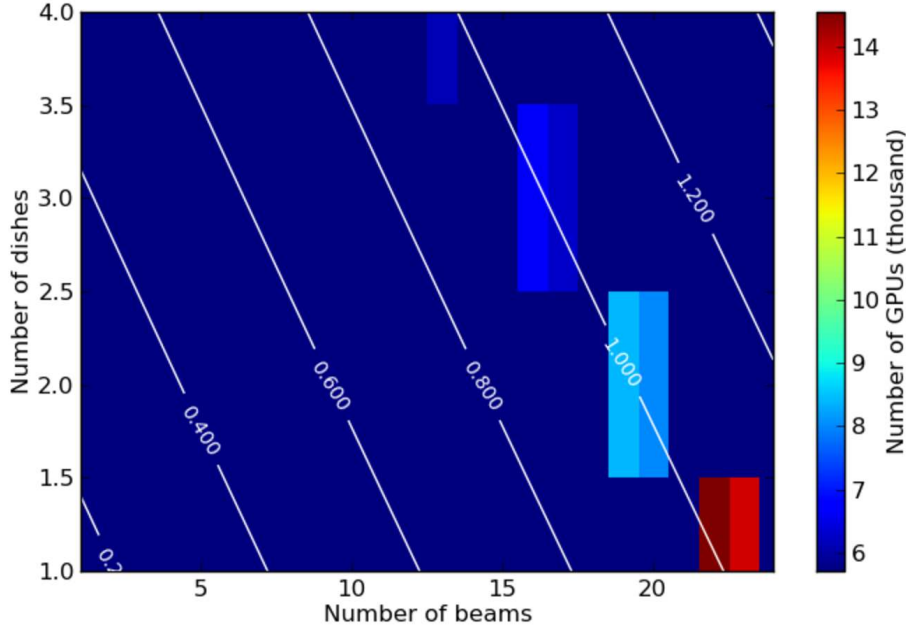


Figure 6.6: Assuming a series of ring-based beamforming chains, where every GPU processes a subset M_a of the dishes, partially synthesising a subset M_b of the output beams, the set of optimal values for M_a and M_b lies along the unit contour line. The color mapping represents values for N_{GPUs} where the GPU utilisation rate is within 1% of the peak.

15,350 dedispersed time series have to be generated for 2222 beams, resulting in an operation count of ~ 10 PFLOP/s when using the direct method, equivalent to about 3,500 GPUs when accounting for e_d , which is clearly unfeasible, indicating that alternative dedispersion methods need to be investigated. Some of these techniques were discussed in section 2.1. [Dewdney, 2013] base their analysis on the Taylor-tree method with piecewise linear approximation, where the frequency band is partitioned into 64 subbands. Following [Barsdell et al., 2012], the maximum DM value which can be processed, referred to as the 'diagonal' DM, with this method is

$$\text{DM}_{\text{diag}} = \frac{N'_c - \frac{1}{2}}{\Delta T(N_c) - \Delta T(N_c - N'_c)} \quad (6.26)$$

$$\Delta T(c) = \frac{k_{\text{DM}}}{\Delta t} ((f_0 + c\Delta f)^{-2} - f_0^{-2}) \quad (6.27)$$

where $\Delta T(c)$ is defined in section 2.1.1 (equation 2.2) and N'_c is the length of each subband. The Taylor-tree method requires the number of channels to be a power

of 2, therefore each subband must be composed of

$$N'_c = 2^{\lceil \log_2 \left(\frac{N_c}{N_{\text{sub}}} \right) \rceil} \quad (6.28)$$

where N_{sub} is the number of frequency partitions. For SKA₁-mid, $N'_c = 256$, therefore an extra 1384 zero-padded frequency channels have to be injected across the band. The total number of frequency channels becomes 16384, while still retaining the original spectral resolution. Together with the temporal resolution specified in table 6.7, $\text{DM}_{\text{diag}} = 73.37 \text{ pc cm}^{-3}$, with a DM step equivalent to $\sim 0.008 \text{ pc cm}^{-3}$.

In order to reach the maximum DM required additional processing would have to be applied, such as time binning, or applying time delays to the beam-formed time series such that the diagonal DM is at a DM of zero at each iteration. The latter option still requires a band-wide, frequency-dependent delay to be applied for band alignment, however it leads to an overall reduction in computational requirements. This will also result in a loss in pulse S/N for each binning iteration. We base our analysis on this technique. The optimal subband size is a balance between computational speedup and additional signal smearing induced by the linear approximation to a quadratic dispersion curve, however for the scope of this section, we will assume 64 subbands, which also corresponds to the peak performance achieved by [Barsdell et al., 2012]. In order to process all DMs up to DM_{max} , the time series has to pass through

$$N_{\text{bins}} = \left\lceil \log_2 \left(\frac{\text{DM}_{\text{max}}}{\text{DM}_{\text{diag}}} \right) \right\rceil + 1 \quad (6.29)$$

binning stages. A total of N'_c DM trials have to be processed in the first stage of each iteration. During the seconds stage, $N'_{\text{DM},i}$ trials need to be processed, where i is the binning stage. This value depends on the maximum DM value associated with each binning stage, $\text{DM}_{\text{max},i}$ and the associated DM step, ΔDM_i

$$N'_{\text{DM},i} = \begin{cases} \text{DM}_{\text{max},i} / \Delta\text{DM}_{\text{max}} & i == 1 \\ (\text{DM}_{\text{max},i} - \text{DM}_{\text{max},i-1}) / \Delta\text{DM}_{\text{max}} & i > 1 \end{cases} \quad (6.30)$$

where $\Delta\text{DM}_{\text{max}} = \max(\Delta\text{DM}_i, \Delta\text{DM})$. Using these relationships and the values from table 6.7, table 6.8 lists the required dedispersion parameters when using the

Binning Stage	1	2	3	4	5
Binning Factor	1	2	4	8	16
Sampling Time (μs)	50	100	200	400	800
Maximum DM (pc cm^{-3})	73.37	146.74	293.48	586.95	1173.90
Equivalent DM Step (pc cm^{-3})	0.008	0.016	0.033	0.65	4 0.131
Number of DMs	1358	1358	2717	4486	1834

Table 6.8: SKA₁-mid survey parameters when using piecewise linear tree dedispersion.

piecewise linear tree algorithm. The operation count for this method is

$$\mathcal{F}_{\text{tree}} = N_b \cdot (\text{stage}_1 + \text{stage}_2) \quad (6.31)$$

$$\text{stage}_1 = \sum_{i=0}^{N_{\text{bin}}-1} \left(8 \frac{N_t}{2^i} N_{\text{sub}} N'_c \log_2(N'_c) \right) \quad (6.32)$$

$$\text{stage}_2 = \sum_{i=0}^{N_{\text{bin}}-1} \left(8 \frac{N_t}{2^i} N_{\text{sub}} N'_{\text{DM},i} \right) \quad (6.33)$$

The computational cost for time binning and delaying is assumed to be negligible relative to the dedispersion cost, and additionally they could also be performed by the CPU. Factors of 8 are included to compensate for delay computation, rounding and other operations. Due to the high algorithmic complexity of this technique, the lack of available optimised implementations and the use of modulus and division operations required during the first stage, an efficiency factor, e_{tree} , of 10% is attributed to the first stage, while the same factor as direct dedispersion is applied to second stage. This yields a value for $\mathcal{F}_{\text{tree}}$ of ~ 78 TFLOP/s, which is equivalent to approximately 26 GPUs when taking arithmetic efficiency into account.

Next we analyse the memory requirements for this method. If the beam-formed time series are binned in place, the piecewise linear algorithm requires

$$M_{\text{tree}} = \text{maxshift} + N_t N_c n_b + \sum_{i=0}^{N_{\text{bins}}-1} \left[\frac{N_t}{2^i} (N_c n_{\text{tree}} + N_{\text{DM},i} n_{\text{DM}}) \right] \quad (6.34)$$

per beam, where maxshift represents the amount of extra memory required to process up to DM_{max} . If these samples are kept in their binned form, then maxshift has a value of $N_c^2 N_{\text{bins}} n_b$. n_{tree} is the bit representation of the output series from the first stage and n_{DM} is the bit representation of the fully dedispersed time series.

n_{tree} should accommodate the temporary summations during the first stage, and, assuming bit representations of powers of 2, should have a value of at least

$$n_{\text{DM}} \geq 2^{\lceil \log_2(\log_2(N'_c \cdot 2^{n_b})) \rceil} \quad (6.35)$$

Equation 6.34 assumes that the intermediary stages for all binning stages, as well as the output dedispersed time series, will be kept in GPU memory. This requirement can be alleviated by overwriting the intermediary stages, and transferring the processed data out of GPU memory after every stage. An additional optimisation is to overwrite the input time series with the output from the second stage, given that they occupy the same amount of memory, otherwise the input buffer would have to be enlarged. Internal GPU bandwidth is of the order of 300 GB/s, so the effect on overall execution time would be negligible. The memory requirement then becomes

$$M_{\text{tree}} = \text{maxshift} + \begin{cases} 2N_t N_c n_{\text{tree}} & N_t N_c n_{\text{tree}} > \beta \\ N_t N_c n_{\text{tree}} + \beta & N_t N_c n_{\text{tree}} < \beta \end{cases} \quad (6.36)$$

$$\beta = \max(N_t N_{\text{DM},0}, \dots, \frac{N_t}{2^{N_{\text{bins}}-1}} N_{\text{DM},N_{\text{bins}}-1}) \quad (6.37)$$

The new size for maxshift will depend on whether the binning process will be performed entirely on the GPU, or on the CPU during the GPU dedispersion stages, with I/O transfer overlapping GPU processing. Here we choose the latter approach, since it requires less memory. Maxshift then simply becomes $N_c^2 n_b$. These optimisations reduce the memory requirements to ~ 1.9 TB, equivalent to approximately 320 GPUs. This value is close to the memory requirement for direct dedispersion, which amounts to approximately 1.2 TB, excluding maxshift.

It is evident that even with the optimisations mentioned above, GPU-based dedispersion for SKA₁-mid will be memory limited. The total input and output data rate is approximately 670 GB/s, which can be accommodated by approximately 50 GPUs. Table 6.9 summarises the GPU requirements described in this section. A rough estimate suggests that almost 400% more GPUs are required to keep the data in memory than those required for I/O and computation.

Processing	78 TFLOP	26 GPUs
I/O transfers	670 GB/s	50 GPUs
Memory	1.7 TB	285 GPUs

Table 6.9: Dedispersion requirements for SKA₁-mid.

6.8 Conclusion

The Square Kilometre Array will pose considerable computational and data transport challenges, some which we have investigated in this chapter, concentrating mostly on station-level processing for SKA₁-low and the Central Signal Processing non-imaging pipeline for SKA₁-mid. The applicability of GPUs for both station-level and CSP beamforming was investigated, implementations of which will be I/O limited, with PCIe transfers being the major bottleneck for GPU-based systems. In the case for CSP beamforming for SKA₁-mid 87% of total execution time would be spent performing I/O, driving the number of GPUs required for beamforming upward of 5,000. The compute-to-transfer time ratio for SKA₁-low station beamforming is higher, however still I/O limited. When considering the running cost for GPU-based systems, especially power consumption costs, it might be more beneficial to deploy an FPGA-based system. GPUs are not well suited for very high I/O scenarios.

We have also analysed the hardware requirements for SKA₁-mid CSP dedispersion, in connection with the non-imaging pipeline, and highlight the fact that this process will be memory-limited, where 400% more GPUs are required to keep the data being processed in memory. This requirement will be even more severe when binary searches are taken into consideration, however the compute requirements will also be much higher, which might tip over the balance between memory requirements and compute requirements.

The analysis performed in this chapter are based on numbers and parameters listed in the SKA Baseline Design [Dewdney, 2013], however the methods described are applicable to any telescope configuration.

CHAPTER 7

CONCLUSION

The primary aim of the work presented in this thesis was to investigate the use and applicability of Graphical Processing (GPUs) for real-time, non-imaging pipelines, with significant emphasis on dedispersion, automatic classification of detected events and beamforming. This has led to the development of a scalable, flexible and high performance software pipeline which has been used to process online data from radio telescopes, such as BEST-II and LOFAR, as well as post-processing of several data products, including GBT observations of Kepler’s field of view, which is still underway.

Dedispersion

The direct dedispersion algorithm was optimised for GPU hardware, based on the work by [Magro et al., 2011] and [Armour et al., 2012], registering a speed-up, compared to an optimised CPU implementation, of more than one order of magnitude, comparable to that reported by [Barsdell et al., 2012]. This leads to a significant reduction in hardware requirements for online systems. Performance benchmarks presented in chapter 3 demonstrate that 8 beams with 20 MHz bandwidth (or alternatively a single beam of 160 MHz, since the number of beams and processable bandwidth are inversely proportional) can be processed with a single server hosting two mid-range GPUs, for up to almost 1000 DM trials. This was compared to similar state-of-the-art systems currently in development, including an FPGA-based approach by [D’Addario et al., 2013], and show that GPUs are a favourable architecture for transient searching.

Direct dedispersion scales linearly with increasing number of beams and DM trials, and we show that for SKA₁-mid approximately 10 PFLOPs would be required to process all the beams, equivalent to $\sim 3,500$ GPUs, this excluding memory

requirements and I/O transfer time. Alternative dedispersion algorithms have to be investigated. Following [Dewdney, 2013], where the Taylor tree algorithm with piecewise linear approximation was selected as a candidate for dedispersion, we provide a detailed analysis of the computational, memory and I/O requirements. We note that with this technique the system will be memory limited, with 400% more GPUs required to keep the data in memory than those required for computation and I/O transfer. For SKA₁-mid non-imaging requirements, approximately 320 GPUs would be required for dedispersion.

Online Transient Detection Pipeline

Fast transient detection pipelines require considerable computational resources, and several processing platforms are being investigated to suite these requirements. In this thesis we propose a GPU-based solution, moulded around a custom GPU framework, whereby data input and buffering is performed by the CPU, with compute intensive tasks offloaded to the GPU. Several processing stages were implemented in this pipeline, including RFI mitigation, dedispersion, event detection and classification, as well as data quantisation and persistence. RFI mitigation through bandpass correction and thresholding removes high power spatial and temporal terrestrial signals which would result in a high number of false detections. The set of dedispersed time series are thresholded and then clustered using a density-based clustering algorithm. The DM-S/N curve of each cluster candidate is then compared to a fitted model which filters out RFI-induced clusters. The detection of an astrophysical event triggers the data persistence module, which quantises and writes the input buffer to disk, containing the event together with cluster parameters. These stages are encapsulated as a standalone framework which can be used in offline mode, for processing archival data, as well as within an online application with additional real-time capabilities. The prototype presented here is capable of processing multiple independent beams in parallel across as many GPUs as are attached to the host.

The accuracy of the RFI mitigation and event detection stages was examined with simulated data sets containing RFI and dispersed pulses which were passed through the pipeline. Narrow pulses, of the order of 2 to 10 samples wide, were only fully detected for a pulse S/N of 4.0 or greater, while wider pulses were detected for pulse S/N as low as 2.0. Most of the simulated RFI signals did pass through the RFI mitigation stage, however 100% of the broadband signals were correctly classified as RFI, whilst a single cluster caused by a narrowband burst

was incorrectly classified as an astrophysical event. This leads to two conclusions:

- Clustering algorithms, coupled with appropriate cluster selection techniques, are a viable candidate for event detection, however more robust candidate selection mechanisms are required to increase the detectability of short duration, low power signals. These include pattern matching and machine learning based techniques. Any supervised learning algorithm will likely require input data sets from several telescope locations, as the RFI environment will change. Ways of generating labelled data sets also need to be investigated.
- Provided that the event detection scheme above is available, any RFI mitigation stage need only filter high-power signals so as to reduce the number of detected data points after dedispersion. Any unmitigated RFI signals will then be filtered out by the candidate classification stage.

This pipeline was also enhanced with real-time capabilities, consisting primarily of high-speed stream processing functionality capable of processing a 5.12 Gb/s SPEAD stream. This system was then deployed on the Medicina BEST-II array, where a ROACH-based digital backend performs all the required pre-processing prior to transmitting beamformed data to the transient detection system. Several test observations were conducted, especially on PSR B0329+54, which is the brightest pulsar observable by BEST-II. Through these observation appropriate threshold parameters were set to various stages in the pipeline, leading to the automatic classification of pulses originating from B0329+54 from RFI-induced events, serving as a online test case for the pipeline.

Beamforming

Signals from multiple antennas have to be combined prior to transient searching (except for cases where an aperture array is used in Fly's Eye mode). Beamforming is a computationally expensive process, with considerable I/O requirements for large N aperture arrays where all the antennas cannot be combined using one processing node. Beamforming implementation tend to be I/O limited. In chapter 5 we presented a GPU implementation of a coherent multi-beam beamformer which can synthesise a number of coherent beams for arrays with an arbitrary number of elements and frequency channels. The kernel utilises 50% of the peak theoretical performance on the test device, at 1.2 TFLOPs, and is limited by shared memory bandwidth. The overall implementation is limited by PCIe bandwidth, where for

a serial execution pipeline, 50% of the processing time is spent transferring data from CPU to GPU memory. If the beams need to be post-processed outside of the GPU on which they were generated, the GPU remains mostly idle, waiting for data transfer requests to terminate.

This discrepancy between computational and I/O requirements can be alleviated by performing additional operations after beam generation, and to test this assertion we have integrated this kernel with the transient detection pipeline described in chapter 3. A mock BEST-II backend running just the F-Engine was set up and used to benchmark this system, and we show that for small, low bandwidth arrays the computational cost for beamforming is negligible when compared to dedispersion. This gives rise to potential systems where these operations are integrated, allowing for dynamic observations where beam pointings can be updated online, in real time, triggered by interesting events during the event detection stage of the transient detection pipeline. Having the raw voltage data available in memory can potentially enable on-the-fly correlation and image generation for more accurate transient source localisation.

We have also presented a test beamforming setup for SKA₁-low station beamforming as well as SKA₁-mid CSP beamforming, based on a ring setup. We demonstrate that GPUs are ill-suited for this task, and the number of processing nodes required depends entirely on the input, output and inter-node bandwidth requirements. For SKA₁-low up to 10 antennas can be combined on a single GPU, and even when channelisation is performed in tandem, the devices spend most of the time waiting for PCIe transfers. This situation is worse for SKA₁-mid CPS, where more than 5,000 GPUs are required, with 87% of execution time spent waiting for I/O transfers. GPUs are also power hungry devices, with a TDP of approximately 250 W. A GPU-based setup would result in very high operational costs, especially when compared to high-end FPGA-based devices with comparable processing capabilities, as demonstrated by [Hickish, 2013].

Extensions and Future Work

The work presented in this thesis led to the development of a pipeline prototype for online fast transient detection which can be deployed on any radio telescope and used to process archival data, and could potentially be used as a reference design for future pipelines, and make possible the creation of a dynamic, high performance, transient search software pipeline through collaborative work. By making the data interfacing mechanisms more generic and providing a cleaner

model by which new processing stages can be implemented and attached to the system, this software can serve as a starting point for creating specialised backends for current and future radio instruments. We believe that GPUs are an ideal candidate for transient detection, and more work needs to be done in this field. Pulsar search software require additional processing stages, such as harmonic summing and binary searches, both of which are computationally expensive. To date, investigation into porting acceleration search algorithms to GPUs has been limited, and although these algorithms would likely be memory limited, the advances in GPU technology will make it possible for the entire pulsar search pipeline to be ported to GPUs. In order to meet the full non-imaging requirements for SKA₁-mid, within a reasonable cost, algorithmic and implementation developments in the coming years will prove to be critical.

REFERENCES

- [Allal et al., 2009] Allal, A. D., Weber, R., Cognard, I., Desvignes, G. and Theureau, G. (2009). RFI Mitigation on the Context of Pulsar Coherent De-Dispersion at the Nancay Radio Astronomical Observatory. In Proceedings of 17th European Signal Processing Conference (EUSIPCO) pp. 2052–2056,.
- [Armour et al., 2012] Armour, W., Karastergiou, A., Giles, M., Williams, C., Magro, A., Zagkouris, K., Roberts, S., Salvini, S., Dulwich, F. and Mort, B. (2012). A GPU-based survey for millisecond radio transients using ARTEMIS. In ASP Conference Series p. 33,.
- [Backer et al., 1997] Backer, D. C., Dexter, M. R., Zepka, A. et al. (1997). Multifrequency polarimetry of 300 radio pulsars. Publications of the Astronomical Society of the Pacific *109*, 61–68.
- [Bannister and Cornwell, 2011] Bannister, K. W. and Cornwell, T. (2011). Two efficient, new techniques for detecting dispersed radio pulses with interferometers: The Chirpolator and The Chimageator. The Astrophysical Journal Supplement Series *196*, 16.
- [Barsdell et al., 2011] Barsdell, B. R., Bailes, M., Barnes, D. G. and Fluke, C. J. (2011). Spotting Radio Transients with the help of GPUs.
- [Barsdell et al., 2010] Barsdell, B. R., Barnes, D. G. and Fluke, C. J. (2010). Analysing Astronomy Algorithms for GPUs and Beyond. Monthly Notices of the Royal Astronomical Society *4096*, 1963.
- [Barsdell et al., 2012] Barsdell, B. R., Bailes, M., Barnes, D. G. and Fluke, C. J. (2012). Accelerating Incoherent Dedispersion. Monthly Notices of the Royal Astronomical Society *422*, 379–392.
- [Bastian, 1994] Bastian, T. S. (1994). Stellar Flares. Space Science Review *68*, 261.
- [Berger et al., 2001] Berger, E., Ball, S., Becker, K. M., Clarke, M., Frail, D. A. et al. (2001). Discovery of radio emission from the brown dwarf LP944-20. Nature *410*, 338–340.
- [Bhat et al., 2013] Bhat, N. D., Chengalur, J. N., Cox, P. J., Gupta, Y., Prasad, J. et al. (2013). Detection of Fast Transients with Radio Interferometric Arrays. The Astrophysical Journal Supplement Series *28*, 2.

- [Bhat et al., 2004] Bhat, N. D. R., Cordes, J. M., Camilo, F., Nice, D. J. and Lorimer, D. R. (2004). Multifrequency Observations of Radio Pulse Broadening and Constraints on Interstellar Electron Density Microstructure. *The Astrophysical Journal* *605*, 759–783.
- [Bhattacharya, 1998] Bhattacharya, D. (1998). 1998 of Radio Emission from Pulsars: a Pulsar Observation Primer. In NATO ASIC Proc. 515: The Many Faces of Neutron Stars.
- [Bloom et al., 2009] Bloom, J. S., Holz, D. E., Hughes, S. A., Menou, K. et al. (2009). Coordinated Science in the Gravitational and Electromagnetic Skies. In *astro2010: The Astronomy and Astrophysics Decadal Survey* vol. 2010, of *Astronomy* p. 20,.
- [Burke and Graham-Smith, 2010] Burke, B. F. and Graham-Smith, F. (2010). An introduction to radio astronomy. Cambridge University Press.
- [Burke-Spolaor, 2012] Burke-Spolaor, B. (2012). Rotating Radio Transients and their place amongst pulsars. In *Neutron Stars and Pulsars: Challenges and Opportunities after 80 years* Proceedings IAU Symposium No. 291.
- [Burke-Spolaor and Bailes, 2010] Burke-Spolaor, S. and Bailes, M. (2010). The millisecond radio sky: transients from a blind single-pulse search. *Monthly Notices of the Royal Astronomical Society* *402*, 855–866.
- [Carilli and Rawlings, 2004] Carilli, C. L. and Rawlings, S. (2004). Science with the Square Kilometre Array, Science with the Square Kilometre Array. *New Astronomy Reviews* *48*, 979–1606.
- [Cenko et al., 2006] Cenko, S. B., Kasliwal, M., Harrison, F. A., Pal’shin, V. et al. (2006). Multiwavelength Observations of GRB 050820A: An Exceptionally Energetic Event Followed from Start to Finish. *The Astrophysical Journal* *652*, 490.
- [Clarke et al., 2013a] Clarke, M. A., La Plante, P. J. and Greenhill, L. J. (2013a). Accelerating radio astronomy cross-correlation with graphics processing units. *International Journal of High Performance Computing Applications* *27*, 178–192.
- [Clarke et al., 2013b] Clarke, N., Macquart, J. P. and Tott, C. (2013b). Performance of a novel fast transients detection system. *The Astrophysical Journal Supplement Series* *205*, 4.
- [Clarke et al., 2011] Clarke, N. L., D’Addario, L., Navarro, R., H., C. T. and Trinh, J. (2011). An Architecture for Incoherent Dedispersion. *CRAFT Memo* 6.
- [Cognard et al., 1996] Cognard, I., Shrauner, J. A., Taylor, J. H. and Thorsett, S. E. (1996). Giant Radio Pulses from a Millisecond Pulsar. *Astrophysical Journal* *457*, L81.

- [Colegate and Clarke, 2011] Colegate, T. M. and Clarke, N. (2011). Searching for Fast Radio Transients with SKA Phase 1. *Publications of the Astronomical Society of Australia* 28, 299–316.
- [Cordes et al., 2006] Cordes, J., Freire, P. C. C., Lorimer, D. R., Camilo, F. et al. (2006). Arecibo Pulsar Survey Using ALFA. I. Survey Strategy and First Discoveries. *The Astrophysical Journal* 637, 446.
- [Cordes, 2009] Cordes, J. M. (2009). The SKA as a Radio Synoptic Survey Telescope: Widefield Surveys for Transients, Pulsars and ETI. *SKA Memo* 97.
- [Cordes et al., 1997] Cordes, J. M., Lazio, T. J. and Sagan, C. (1997). Scintillation-induced Intermittency in SETI. *The Astrophysical Journal* 487, 782.
- [Cordes and Lazio, 2002] Cordes, J. M. and Lazio, T. J. W. (2002). NE2001. I. A New model for the galactic distribution of free electrons and its fluctuations.
- [Cordes and McLaughlin, 2003] Cordes, J. M. and McLaughlin, M. A. (2003). Searches for Fast Radio Transients. *Astrophysical Journal* 593, 1142.
- [D’Addario, 2010] D’Addario, L. (2010). Searching for Dispersed Transient Pulses with ASKAP. *SKA Memo* 124.
- [D’Addario et al., 2013] D’Addario, L., Clarke, N., Navarro, R. and Trinh, J. (2013). An FPGA-based back end for real time, multi-beam transient searches over a wide dispersion measure range. In Presented in Radio Science Meeting (USNC-URSI NRSM), 2013 US National Committee of URSI National.
- [Deneva et al., 2009] Deneva, J. S., Cordes, J. M., McLaughlin, M. A., Nice, D. J. et al. (2009). Arecibo Pulsar Survey Using ALFA: Probing Radio Pulsar Intermittency And Transients. *The Astrophysical Journal* 703, 2259–2274.
- [Dewdney, 2013] Dewdney, P. E. (2013). SKA1 System Baseline Design. .
- [Dewdney et al., 2010] Dewdney, P. E., bij de Vaate, J., Cloete, K., Gunst, A., Hall, D. et al. (2010). SKA Phase 1: Preliminary System Description. *SKA Memo* 130.
- [Dingus et al., 2002] Dingus, B., Laird, R. and Sinnis, G. (2002). Search for GeV Gamma Rays from the Quark-Gluon Plasma Surrounding Evaporating Primordial Black Holes. vol. 34, of COSPAR Meeting.
- [Dulk, 1985] Dulk, G. A. (1985). Radio Emission from the Sun and Stars. *Annual Review of Astronomy and Astrophysics* 23, 169.
- [Ester et al., 1996] Ester, M., Kriegel, H., Jörg, S. and Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise . In Second International Conference on Knowledge Discovery and Data Mining pp. 226–231,.

- [Fender, 2012] Fender, R. (2012). The scientific potential of LOFAR for time domain astronomy. In *Proceedings of the International Astronomical Union* vol. 7, pp. 11–16,.
- [Foster, 2013] Foster, G. S. (2013). Large-N Correlator Systems for Low Frequency Radio Astronomy. PhD thesis, University of Oxford.
- [Fridman, 2010] Fridman, P. A. (2010). A method of detecting radio transients. *Monthly Notices of the Royal Astronomical Society* *409*, 808–820.
- [Gaensler et al., 2013] Gaensler, M. A. et al. (2013). Key Science Projects for the SKA. SKA Memo 44.
- [Gould and Lyne, 1998] Gould, D. M. and Lyne, A. G. (1998). Multifrequency polarimetry of 300 radio pulsars. *Monthly Notices of the Royal Astronomical Society* *301*, 235–260.
- [Güdel, 2002] Güdel, M. (2002). Stellar Radio Astronomy: Probing Stellar Atmospheres from Protostars to Giants. *Annual Review of Astronomy and Astrophysics* *596*, 982–996.
- [Hallinan et al., 2007] Hallinan, G., Bourke, S., Lane, C., Antonova, A., Zavala, R. T. et al. (2007). Periodic Bursts of Coherent Radio Emission from an Ultra-cool Dwarf. *The Astrophysical Journal* *663*, L25–L28.
- [Hankins et al., 2003] Hankins, T. H., Kern, J. S., Weatherall, J. C. and Eilek, J. A. (2003). Nanosecond radio bursts from strong plasma turbulence in the Crab pulsar. *Nature* *422*, 141–143.
- [Hansen and Lyutikov, 2001] Hansen, B. M. and Lyutikov, M. (2001). Radio and X-ray signatures of merging neutron stars. *Monthly Notices of the Royal Astronomical Society* *322*, 695–701.
- [Hassall et al., 2013] Hassall, T. E., Keane, E. F. and Fender, R. P. (2013). Detecting highly-dispersed bursts with next-generation radio telescopes.
- [Hessels et al., 2008] Hessels, J. W., Stappers, B. W., van Leeuwen, J. and LOFAR Transients Key Science Project (2008). The Radio Sky on Short Timescales with LOFAR: Pulsars and Fast Transients. In *Proceedings of "The Low-Frequency Radio Universe"*.
- [Hewish et al., 1968] Hewish, A., Bell, S. J., Pilkington, J. D. H., Scott, P. F. and Collins, R. A. (1968). Observation of a Rapidly Pulsating Radio Source. *Nature* *217*, 709–713.
- [Hickish, 2013] Hickish, J. (2013). Digital Signal Processing Methods in Large, Distributed, Low-Frequency Radio Telescopes. PhD thesis, University of Oxford.
- [Hobbs et al., 2003] Hobbs, G., Manchester, R., Teoh, A. and Hobbs, M. (2003). The ATNF Pulsar Catalogue. In *Young Neutron Stars and their Environments* vol. 218, of *Proceedings of IAU Symposium*.

- [Hodgen et al., 2012] Hodgen, J., Wiel, S., Bower, G., Siemion, A. and Werthimer, D. (2012). Comparison of RFI Mitigation Strategies for Dispersed Pulse Detection.
- [Hyman et al., 2006] Hyman, S. D., Lazio, T. J. W., Roy, S., Ray, P. S. and Kasim, N. E. (2006). A Faint, Steep Spectrum Burst from the Radio Transient GCRT J17453009. *Astrophysical Journal* *639*, 348.
- [Jansky, 1933] Jansky, K. (1933). Electrical Disturbances Apparently of Extraterrestrial Origin. *Proceedings of the Institute of Radio Engineers* *21*, 1387–1398.
- [Johnson, 1928] Johnson, J. (1928). Thermal Agitation of Electricity in Conductors. *Physical Review* *32*, 97–109.
- [Johnston and Romani, 2003] Johnston, S. and Romani, W. (2003). Giant Pulses from PSR B0540-69 in the Large Magellanic Cloud. *The Astrophysical Journal Letters* *590*, L95.
- [Keane et al., 2011] Keane, E. F., Kramer, M., Lyne, A. G., Stappers, B. W. and McLaughlin, M. A. (2011). RRATs: New Discoveries, Timing Solutions & Musings. *Monthly Notices of the Royal Astronomical Society* *415*.
- [Keane et al., 2012] Keane, E. F., Stappers, B. W., Kramer, M. and Lyne, A. G. (2012). On the Origin of a Highly Dispersed Coherent Radio Burst. *Monthly Notices of the Royal Astronomical Society* *425*, L71–L75.
- [Keith et al., 2012] Keith, M. J., Jameson, A., W., v., Bailes, S. et al. (2012). The High Time Resolution Universe Pulsar Survey I. System configuration and initial discoveries. *Monthly Notices of the Royal Astronomical Society* *409*, 619–627.
- [Klebesadel et al., 1973] Klebesadel, R. W., Strong, I. and Olson, R. (1973). Observations of Gamma-Ray Bursts of Cosmic Origin. *Astrophys. J. Lett.* *182*, L85–L88.
- [Kondo et al., 2009] Kondo, H., Heien, E., Okita, M., Werthimer, D. and Hagi-hara, K. (2009). A Multi-GPU Spectrometer System for Real-Time Wide Bandwidth Radio Signal Analysis. In *International Symposium on Parallel and Distributed Processing with Applications* pp. 594–604,.
- [Law and Bower, 2012] Law, C. J. and Bower, G. C. (2012). All transients, all the time: real-time radio transient detection with interferometric closure quantities. *The Astrophysical Journal* *749*, 143.
- [Law et al., 2011] Law, C. J., Jones, G., Backer, D. C., Barott, W. C. et al. (2011). Millisecond imaging of radio transients with the pocket correlator. *The Astrophysical Journal* *742*, 12.

- [Linton et al., 2006] Linton, E. T., Atkins, R. W., Badran, H. M., Blaylock, G., Boyle, P. J. et al. (2006). A new search for primordial black hole evaporations using the Whipple gamma-ray telescope. *Journal of Cosmology and Astroparticle Physics* *2006*, 333–334.
- [Lorimer et al., 2007] Lorimer, D., Bailes, M., McLaughlin, M., Narkevic, D. and Crawford, F. (2007). A Bright Millisecond Radio Burst of Extragalactic Origin. *Science* *318*, 777–780.
- [Lorimer, 2006] Lorimer, D. R. (2006). SIGPROC-v3.7 : (Pulsar) Signal Processing Programs. Technical report.
- [Lorimer et al., 2013] Lorimer, D. R., Karastergiou, A., McLaughlin, M. A. and Johnston, S. (2013). On the detectability of extragalactic fast radio transients.
- [Lorimer and Kramer, 2005] Lorimer, D. R. and Kramer, M. (2005). *Handbook of Pulsar Astronomy*. Cambridge University Press.
- [Macquart, 2011] Macquart, J. P. (2011). Detection Rates for Surveys for Fast Transients with Next Generation Radio Arrays. *The Astrophysical Journal* *734*, 20.
- [Macquart et al., 2010] Macquart, J. P., Bailes, M., Bhat, N. D. R., Bower, C. G. et al. (2010). The Commensal Real-Time ASKAP Fast-Transients (CRAFT) Survey. *Publications of the Astronomical Society of Australia* *27*, 272–282.
- [Magro et al., 2011] Magro, A., Karastergiou, A., Salvini, S., Mort, B., Dulwich, F. and Zarb Adami, K. (2011). Real-time, fast radio transient searches with GPU de-dispersion. *Monthly Notices of the Royal Astronomical Society* *17*, 2642–2650.
- [Manchester et al., 1996] Manchester, R. N., Lyne, A. G., D’Amico, N., Bailes, M. et al. (1996). The Parkes southern pulsar survey - 1. Observing and data analysis systems and initial results. *Monthly Notices of the Royal Astronomical Society* *279*, 1235–1250.
- [Manley et al., 2012] Manley, J., Welz, M., Parsons, A., Ratcliffe, S. and van Rooyen, R. (2012). SPEAD Recommended Practise. SKA South Africa Document.
- [McLaughlin and Cordes, 2003] McLaughlin, M. A. and Cordes, J. M. (2003). Searches for Giant Pulses from Extragalactic Pulsars. *The Astrophysical Journal* *596*, 982–996.
- [McLaughlin et al., 2006] McLaughlin, M. A., Lyne, A. G., Lorimer, D. R., Kramer, M. et al. (2006). Transient radio bursts from rotating neutron stars. *Nature* *439*, 817–820.

- [Montebugnoli et al., 2009] Montebugnoli, S., Bianchi, G., Monari, J., Nalid, J., Perini, F. and Schiaffino, M. (2009). BEST: Basic Element for SKA Training. In *Proceedings of Wide Field Astronomy & Technology for the Square Kilometer Array (SKADS 2009)* pp. 331–336,.
- [NVIDIA, Corporation, 2012] NVIDIA, Corporation (2012). NVIDIA's Next Generation CUDA™ Compute Architecture: Kepler™ GK110. White paper.
- [Osten and Bastian, 2008] Osten, R. A. and Bastian, T. S. (2008). Ultrahigh Time Resolution Observations of Radio Bursts on AD Leonis. *Astrophysical Journal* *674*, 1078.
- [Prasad and Wijnholds, 2012] Prasad, P. and Wijnholds, S. J. (2012). AART-FAAC: Towards a 24x7, All-sky Monitor for LOFAR.
- [Ransom, 2001] Ransom, S. (2001). New Search Techniques for Binary Pulsars. PhD thesis, Harvard University.
- [Ransom et al., 2009] Ransom, S. M., Demorest, P., J., F., McCullough, R., Ray, J., DuPlain, R. and Brandt, P. (2009). GUPPI: Green Bank Ultimate Pulsar Processing Instrument. In *American Astronomical Society Meeting Abstracts* vol. 214, p. 605,.
- [Rees, 1977] Rees, M. J. (1977). A better way of searching for black-hole explosions? *Nature* *266*, 333–334.
- [Richards et al., 2003] Richards, M. T., Waltman, E. B., D., G. F. and Richards, D. S. P. (2003). Statistical Analysis of 5 Year Continuous Radio Flare Data from beta Persei, V711 Tauri, delta Librae, and UX Arietis. *ApJS* *147*, 337.
- [Romani and Johnston, 2001] Romani, W. and Johnston, S. (2001). Giant Pulses from the Millisecond Pulsar B182124. *The Astrophysical Journal Letters* *557*, L93.
- [Roy et al., 2009] Roy, J., Gupta, Y., Pen, U., Peterson, J. B. et al. (2009). A real-time software backend for the GMRT. *Experimental Astronomy* *28*, 25–60.
- [Sclocco et al., 2012] Sclocco, A., Varbanescu, A. L., Mol, J. D. and van Nieuwpoort, R. (2012). Radio Astronomy Beam Forming on Many-Core Architectures. In *International Parallel and Distributed Processing Symposium (IPDPS)*.
- [Serylak et al., 2012] Serylak, M., Karastergiou, A., Williams, C., Armour, W., Giles, M. and Group, L. P. W. (2012). Observations of transients and pulsars with LOFAR international stations and the ARTEMIS backend. In *Neutron Stars and Pulsars: Challenges and Opportunities after 80 years* vol. 291,.
- [Shapiro and Teukolsky, 1983] Shapiro, S. L. and Teukolsky, S. A. (1983). *Black Holes, White Dwarfs and Neutron Stars. The Physics of Compact Objects.* Wiley-Interscience.

- [Shkolnik et al., 2005] Shkolnik, E., Walker, G. A., Bohlender, D. A., Gu, P. and Kürster, M. (2005). Hot Jupiters and Hot Spots: The Short- and Long-Term Chromospheric Activity on Stars with Giant Planets. *The Astrophysical Journal* *622*, 1075–1090.
- [Siemion et al., 2012] Siemion, A. P., Bower, G. C., Foster, G., McMahon, P. L., Wagner, M. I., Werthimer, D., Backer, D., Cordes, J. and van Leeuwen, J. (2012). The Allen Telescope Array fly’s eye survey for fast radio transients. *The Astrophysical Journal* *744*, 109.
- [Smits et al., 2008] Smits, R., Kramer, M., Stappers, B., Lorimer, D. R., Cordes, J. and Faulkner, A. (2008). Pulsar searches and timing with the SKA. *SKA Memo* 105.
- [Stappers et al., 2004] Stappers, B. W., H. J. W., Alexov, A., Anderson, A. et al. (2004). Observing pulsars and fast transients with LOFAR. *Astronomy and Astrophysics* *530*, A80.
- [Stars et al., 2002] Stars, I. H., Splaver, E. M., Thorsett, S. E., Nice, D. J. and Taylor, J. H. (2002). A baseband recorder for radio pulsar observations. *Monthly Notices of the Royal Astronomical Society* *314*, 459–467.
- [Taylor, 1974] Taylor, J. H. (1974). A Sensitive Method for Detecting Dispersed Radio Emission. *Reports on Progress in Physics* *15*, 367.
- [Thornton et al., 2013] Thornton, D., Stappers, B., Bailes, B. et al. (2013). A Population of Fast Radio Bursts at Cosmological Distances. *Science* *341*, 53–56.
- [van der Veldt, 2011] van der Veldt, K. (2011). A Polyphase Filter For GPUs And Multi-Core Processors. Master’s thesis Universiteit van Amsterdam.
- [van Haarlem et al., 2012] van Haarlem, M. P., Wise, M. W., Gunst, A. W., Heald, G. et al. (2012). Lofar: The low-frequency array.
- [van Leeuwen and Stappers, 2010] van Leeuwen, J. and Stappers, B. (2010). Finding pulsars with LOFAR. *Astronomy and Astrophysics* *509*, A7.
- [van Straten and Bailes, 2011] van Straten, W. and Bailes, M. (2011). DSPSR: Digital Signal Processing Software for Pulsar Astronomy. *The Astronomical Society of Australia* *28*, 1–14.
- [von Korff, 2010] von Korff, J. S. (2010). Astropulse: A search for microsecond transient radio signals using distributed computing. PhD thesis, University of California, Berkeley.
- [Voûte et al., 2002] Voûte, J. L., Kouwenhoven, M. L., van Haren, P. C. et al. (2002). PuMa, a digital Pulsar Machine. *Astronomy and Astrophysics* *385*, 733–742.

- [Wyath et al., 2011] Wyath, R. B., Briske, W. F., Deller, A. T., Majid, W. A. et al. (2011). V-FASTR: The VLBA Fast Radio Transients Experiment. *The Astrophysical Journal* *735*, 97.
- [Zarb-Adami et al., 2013] Zarb-Adami, K., Bunton, J., Carlson, B., Comoretto, G. et al. (2013). SKA1-Low Hybrid PIP. SKA Documentation.
- [Zarka, 1998] Zarka, P. (1998). Auroral radio emissions at the outer planets: Observations and theories. *Journal of Geophysical Research* *20*, 159.
- [Zarka, 2007] Zarka, P. (2007). Plasma interactions of exoplanets with their parent star and associated radio emissions. *Planetary and Space Science* *55*, 598–617.
- [Zhou et al., 2000] Zhou, S., Zhou, A., Jin, W. and Qian, W. (2000). FDBSCAN: A Fast DBSCAN Algorithm. *Journal of Software* *11*, 735–744.

CHAPTER 8

SKA₁-LOW STATION TRANSIENT PARAMETERS

The table below lists the parameters for SKA₁-low station-level transient searching, discussed in section 6.5, where the station bandwidth is split into 16 frequency subbands, each processed separately on a GPU processing node.

Subband	1	2	3	4	5	6	7	8
Low Frequency (MHz)	50	65.625	81.25	96.87	112.5	128.125	143.75	159.375
High Frequency (MHz)	65.625	81.25	96.87	112.5	128.125	143.75	159.375	175
Maximum DM (pc cm^{-3})	510.18	620.18	722.16	817.74	908.05	993.81	1075.95	1154.64
DM Step (pc cm^{-3})	0.000125	0.000285	0.000535	0.000915	0.001435		0.003005	0.004095
Processable DMs	12016	12016	12016	12016	12016	12016	12016	12016
Number of Channels	15625	15625	15625	15625	15625	15625	15625	15625
Processable Max (pc cm^{-3})	1.50	3.42	6.43	11.00	17.24	25.54	36.11	49.21
DM Range Fraction (%)	0.003	0.006	0.009	0.013	0.019	0.026	0.034	0.043
#Stations Required	340	182	113	75	53	39	30	24

Subband	9	10	11	12	13	14	15	16
Low Frequency (MHz)	175	190.625	206.25	221.875	237.5	253.125	268.75	284.375
High Frequency (MHz)	190.625	206.25	221.875	237.5	253.125	268.75	284.375	300
Maximum DM (pc cm^{-3})	1230.39	1303.51	1374.26	1442.86	1509.50	1574.35	1637.54	1699.19
DM Step (pc cm^{-3})	0.005416	0.007006	0.008876	0.011056	0.013556	0.016417	0.019647	0.023277
Processable DMs	12016	12016	12016	12016	12016	11950	11950	11950
Number of Channels	15625	15625	15625	15625	15625	8192	8192	8192
Processable Max (pc cm^{-3})	65.08	84.19	106.66	132.86	162.90	196.19	234.80	278.19
DM Range Fraction (%)	0.053	0.065	0.078	0.092	0.108	0.125	0.143	0.164
#Stations Required	19	16	13	11	10	9	7	7

Table 8.1: SKA₁-low station fast transient survey parameters